UNIVERSIDADE DA BEIRA INTERIOR
Engenharia Informática

Blockchain-based Decentralized Application for Electronic Voting using an Electronic ID

José Diogo Soares Albergaria Serejo Monteiro

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio
Co-orientador: Eng. Celso Martinho (Bright Pixel)

Covilhã, Outubro de 2019

Acknowledgments

First of all, I would like to thank my parents for their boundless love, support and trust. It would have been unimaginable to have reached this point of my academic life without their guidance.

Second, I deeply thank my supervisor, Professor Doutor Pedro Inácio, for the ceaseless support, encouragement and insight. It was a privilege to work alongside such a brilliant and mesmerizing tutor.

Next, I sincerely acknowledge the assistance of Bernardo Sequeiros and Tiago Simões in reviewing this document.

Lastly, I would like to express deep gratitude to all the friends that have accompanied me through this journey.

Resumo

Um sistema de votação eletrónica que reproduza eficazmente sistemas eleitorais utilizados no mundo é há muito desejado. Até recentemente, não era possível satisfazer, simultaneamente, as propriedades intrínsecas de um sistema de votação tradicional. Recentemente, com o advento de novas tecnologias e investigação, não só é possível cumprir estas propriedades, como também melhorar o anonimato, acessibilidade e a própria estrutura do processo eleitoral.

Neste trabalho, apresentamos um protocolo de votação eletrónica descentralizada e auto contável, que aumenta a privacidade dos eleitores e diminui a centralização. Estas propriedades são alcançadas através de uma relação simbiótica entre a *Blockchain* de *Ethereum* e o cartão de cidadão eletrónico Português. Ao contrário de protocolos eleitorais de recurso a *Blockchain* propostos anteriormente, esta é a primeira implementação que de mais perto atende à maioria dos requisitos de segurança de um esquema de votação real. Além disso, este sistema aperfeiçoa os sistemas de votação eletrónica utilizados atualmente, através da utilização de um protocolo auto contável. Desta forma, cada eleitor é capaz de fazer a contagem eleitoral por si mesmo, e verificar que todos os intervenientes agem de acordo com o protocolo. A execução do protocolo é compelida através do mesmo mecanismo de consenso distribuído que protege a *Blockchain* de *Ethereum*.

Por forma a provar a sua viabilidade, a implementação foi testada na rede oficial de testes *Proof of Work* (PoW) de *Ethereum (Ropsten)*. Os custos financeiros e computacionais estão ao mesmo nível do mais importante protocolo de e-voting associado a *Blockchain*.

Palavras-chave

Aplicações Descentralizadas, Blockchain, Contratos Inteligentes, Ethereum, Protocolos de e-Voting

Resumo alargado

Este capítulo tem por objetivo apresentar, de forma mais detalhada relativamente à secção anterior, um resumo do corpo desta dissertação. Esta dissertação encontra-se escrita, maioritariamente, na Língua inglesa, sendo que este capítulo em concreto se encontra redigido em Português.

Introdução

O primeiro capítulo começa por enquadrar o trabalho descrito nesta dissertação, através da introdução do tema geral e expondo a motivação no âmbito em que este se insere. Neste capítulo é também apresentado o problema a tratar e os objetivos gerais a alcançar, bem como a abordagem adotada para resolver o problema. Por último, são enumerados os contributos decorrentes da investigação e do desenvolvimento deste trabalho.

Enquadramento, Descrição do Problema e Objetivos

Um sistema de voto eletrónico que simule eficazmente sistemas de voto tradicionais é algo bastante desejado atualmente. Até há poucos anos, não era possível satisfazer, simultaneamente, as propriedades indispensáveis a um sistema de votação real, como a segurança, o anonimato, a resistência à coerção e a antirastreabilidade. Contudo, com o advento de novas tecnologias e investigação passou a ser possível garantir o cumprimento destas propriedades.

O objetivo deste trabalho é o de transformar e migrar um sistema físico, nomeadamente um sistema eleitoral, utilizado na sociedade para o mundo digital, e melhorá-lo. Os atuais sistemas de voto eletrónico, apesar de satisfazerem a maior parte dos requisitos físicos de sistema e respetivas funcionalidades, contêm bastantes problemas e vulnerabilidades por superar, resultando num sistema que não emula na totalidade um sistema de voto tradicional. É, pois, o nosso objetivo principal, propor e criar um sistema de voto eletrónico que responda a todos os requisitos de sistema e garanta a segurança de um sistema eleitoral físico.

Desta forma, este projeto de Mestrado visa três objetivos principais: (i) estudar os desafios de segurança relativos a sistemas eleitorais; identificar requisitos (tais como requisitos de segurança); (ii) engenhar um sistema de voto eletrónico; implementar um conjunto de contratos inteligentes de *Ethereum* subjacentes ao caso de estudo descrito neste projeto; (iii) implementar uma aplicação *web* descentralizada que faça uso do Cartão de Cidadão Português, para propósitos relacionados com autenticação, o que permitirá a um eleitor registar um voto, sem a necessidade de possuir conhecimentos relacionados com o sistema subjacente.

Abordagem Adotada para Resolver o Problema

Esta secção apresenta a abordagem tomada para resolver o problema. Este trabalho foi dividido pelas seguintes etapas: (i) estudar os diferentes sistemas de voto físico; (ii) descrever requisitos de sistema do ponto de vista dos requisitos que um sistema de voto eletrónico deverá abranger; (iii) investigar e examinar as diferentes tecnologias necessárias para a implementação do sistema; (iv) desenhar o sistema, tendo em conta os diferentes requisitos de sistema; (v) testar o sistema; (vi) provar o cumprimento das propriedades de segurança; (vii) analisar o custo financeiro do sistema.

Principais Contribuições

Esta secção visa apresentar as principais contribuições científicas resultantes deste trabalho. As principais contribuições são as seguintes:

1. Identificação dos requisitos de segurança de um sistema de voto eletrónico, bem como o desenho de um sistema que responda a esses mesmo requisitos;
2. O desenho, montagem e teste de uma aplicação descentralizada (Dapp) que interaja com contratos inteligentes de Ethereum, relativos a um sistema de voto eletrónico que utilize um cartão de cidadão eletrónico para efeitos de autenticação;
3. O desenho, montagem e teste de uma ponte de comunicação entre a Decentralized Application (Dapp) e o *middleware* do Cartão de Cidadão Português;
4. O desenho, montagem e teste de um servidor de validação da autenticação e elegibilidade do eleitor, bem como dos certificados presentes no cartão de cidadão;
5. Verificação do cumprimento dos requisitos de segurança, teste e análise financeira do sistema.
6. O sistema desenvolvido foi assunto de um artigo científico com o título *Blockchain-based Decentralized Application for Electronic Voting Using an Electronic ID*, aceite para publicação nas respetivas atas e apresentado no 11.º Simpósio de Informática (INForum 2019), que se realizou na Universidade do Minho, nos dias 5 e 6 de setembro de 2019 [1].

Background e Trabalho Relacionado

O segundo capítulo explora os diferentes tópicos que serviram de alicerce a este projeto de Mestrado, tendo sido extensivamente investigados, juntamente com diversos trabalhos científicos que se podem considerar profundamente relacionados com este.

Os quatro tópicos investigados que tiveram um impacto elevado neste trabalho foram os seguintes: (i) *Ethereum* e aplicações descentralizadas, (ii) contratos inteligentes, (iii) sistemas de voto eletrónico e (iv) provas de conhecimento-zero.

Destacam-se alguns trabalhos acerca de sistemas de voto eletrónico descentralizado, tanto ao nível protocolar, conforme apresentado por Hao *et. al.* [2], como ao nível da conceção do próprio sistema, como o *Follow My Vote* [3] ou o *Open Vote Network* [4].

Os diferentes sistemas de voto eletrónico investigados foram sujeitos a comparações relativamente ao cumprimento — ou não — de diferentes requisitos alusivos a sistemas de voto eletrónico, bem como à análise das diferentes tecnologias empregues por estes sistemas. Estas comparações, representadas nas tabelas 2.1 e 2.2, permitiram depreender as vantagens e desvantagens de cada um destes sistemas.

Requisitos e Desenho do Sistema

O terceiro capítulo contempla os diferentes requisitos de segurança relacionados com sistemas de voto eletrónico, bem como especificações particulares relativas a sistemas de voto eletrónico que utilizem contratos inteligentes. Foi também feita uma análise e consequente modelação dos requisitos do sistema de software.

De acordo com R. Anane [5], um sistema de e-voting deve dispor das seguintes propriedades: (i) precisão, isto é, não deverá ser possível alterar um voto, (ii) privacidade, ou seja, um eleitor não poderá ser ligado ao seu voto, (iii) elegibilidade, isto é, apenas utilizadores elegíveis poderão votar, sendo que apenas o poderão fazer uma vez, (iv) verificabilidade individual e universal, ou seja, qualquer eleitor poderá verificar que os votos foram introduzidos e contados corretamente pelo sistema, e (v) justiça, isto é, os resultados de uma eleição não poderão ser revelados até ao *terminus* do período alocado para a mesma. Estas propriedades não são, por vezes e na totalidade, satisfeitas por sistemas de voto tradicionais e eletrónico, devido à centralização inerente ao próprio sistema. Um sistema de voto eletrónico que faça uso de contratos inteligentes é capaz de responder às propriedades acima enunciadas, devido à sua natureza descentralizada, o que origina um sistema mais justo, robusto e honesto.

Os requisitos do sistema de software foram concebidos através da elaboração de diferentes casos de uso, tendo em conta a tipologia de utilizador do sistema (eleitor e administrador da eleição).

Posto isto, foi possível conceber uma proposta para o sistema descrito neste documento. O sistema proposto é composto por um conjunto de programas interconectados, tal como esquematizado na figura 3.4. Em primeiro lugar, a parte mais central do sistema é a aplicação *web*, que é a única parte do sistema com que o utilizador interage diretamente. A aplicação *web* comunica diretamente com dois servidores: (i) um servidor remoto responsável por validar as credenciais do eleitor e (ii) um servidor local com o objetivo de retransmitir as comunicações entre a aplicação *web* e um cliente local. Este cliente, em particular, permite interagir com o cartão de cidadão do eleitor. Tal como descrito anteriormente, a aplicação *web* é descentralizada, uma vez que é capaz de comunicar com contratos inteligentes de *Ethereum*, onde a lógica associada ao protocolo eleitoral se encontra armazenada e onde pode ser publicamente escrutinada.

Implementação

O quarto capítulo contempla os detalhes da implementação dos diversos componentes do sistema de e-voting proposto no capítulo três. Aqui se inclui a abordagem tomada para desenvolver a aplicação *web* descentralizada, o sistema de autenticação com o cartão de cidadão Português e os contratos inteligentes de *Ethereum*.

Conforme descrito anteriormente, a aplicação *web* pode ser considerada como o componente mais central do sistema, sendo responsável pela interação com os restantes componentes. Esta aplicação *web*, desenvolvida em React.JS, contempla dois fluxos de execução distintos: (i) o fluxo do administrador e (ii) o fluxo do eleitor. Ambos os fluxos permitem uma utilização sem um conhecimento aprofundado do sistema subjacente. Por um lado, o fluxo do administrador é responsável por permitir uma gestão simples de todo o processo eleitoral. Por outro, o fluxo do eleitor permite um *onboarding* expedito, com uma rápida progressão para o ato de voto, assim como uma análise compreensiva e transparente dos resultados finais. Ao contrário de aplicações *web* tradicionais, a lógica do protocolo eleitoral da aplicação desenvolvida encontra-se representada num par de contratos inteligentes, os quais podem ser publicamente escrutinados. A aplicação interage com estes contratos através da biblioteca *web*3.JS e da extensão de *browser Metamask*.

O primeiro destes contratos inteligentes é denominado `Cryptography Contract`, sendo responsável por unificar as funções criptográficas utilizadas durante o processo eleitoral, pelos diversos eleitores e pelo administrador da eleição. O segundo contrato é intitulado `Voting Contract`, encarregue de alternar entre as diferentes fases da eleição, verificar que os requisitos estabelecidos pelo administrador são cumpridos, assim como executar algumas das funções do `Cryptography Contract` responsáveis pela verificação das chaves de voto de um determinado eleitor e pelo cálculo do resultado final da eleição.

Cumprimento dos Requisitos de Segurança

O capítulo cinco começa por examinar o cumprimento dos requisitos de segurança referidos no capítulo três, seguindo-se a criação e posterior análise de testes de usabilidade. Por último, é feita uma análise ao custo financeiro do sistema.

As diversas propriedades, extrapoladas dos requisitos de segurança anteriormente referidos, foram satisfeitas na sua totalidade. A propriedade de privacidade é validada através da utilização do cartão de cidadão Português para a validação da elegibilidade e autenticidade de um eleitor, através da sua assinatura digital. A propriedade de precisão é confirmada por meio da imutabilidade da própria *blockchain* de *Ethereum* e da lógica estabelecida nos contratos inteligentes utilizados pela Dapp. A propriedade de justiça é satisfeita, pois só é possível — através da lógica presente no contrato inteligente — calcular o resultado final da eleição quando o prazo do período alocado para a votação terminar. A propriedade de elegibilidade é validada através da utilização das listas de revogação de certificados do cartão de cidadão, o que implica que um cidadão pode apenas dispor de um único cartão de cidadão válido. Deste modo, um eleitor

é apenas capaz de votar uma única vez. A propriedade de verificabilidade individual e universal é confirmada, adotando o protocolo eleitoral descrito por Hao *et. al.* [2], que permite que um eleitor seja capaz de verificar que o seu voto foi registado corretamente, bem como corroborar que os restantes eleitores agem de acordo com o protocolo estabelecido.

Foram criados vários testes específicos, relativamente à usabilidade da aplicação de voto eletrónico. Estes testes foram realizados a um grupo de dez pessoas, de diferentes idades, género e formação académica. Após a análise dos dados obtidos pelos diferentes testes, foi possível concluir que a experiência de usabilidade dos diversos participantes foi predominantemente positiva.

Conclusões

O sétimo capítulo elenca as principais conclusões a serem retiradas deste trabalho e apresenta potencial trabalho futuro a ser realizado.

Este trabalho foi inicialmente dividido em três objetivos principais: (i) estudar os desafios de segurança relativos a sistemas eleitorais; identificar requisitos (tais como requisitos de segurança); (ii) engenhar um sistema de voto eletrónico; implementar um conjunto de contratos inteligentes de *Ethereum* subjacentes ao caso de estudo descrito neste projeto; (iii) implementar uma aplicação *web* descentralizada que faça uso do Cartão de Cidadão Português, para propósitos relacionados com autenticação, o que permitirá a um eleitor registar um voto, sem a necessidade de possuir conhecimentos relacionados com o sistema subjacente. O primeiro objetivo foi satisfeito através do estudo, especificação e esclarecimento dos diferentes requisitos pertencentes a um sistema de voto eletrónico, retratado no terceiro capítulo deste documento. O segundo objetivo foi cumprido mediante a identificação dos requisitos do sistema de software, seguido da elaboração de uma proposta para o sistema, conforme descrito no capítulo terceiro. O terceiro, e último, objetivo foi realizado através da implementação da proposta do sistema, tal como retratado no capítulo quatro.

Apesar dos objetivos principais deste trabalho terem sido cumpridos, existem certas funcionalidades que podem ser acrescentadas ao sistema, com o objetivo de o dotar ainda de uma maior segurança, conveniência e desempenho. Estas funcionalidades são as seguintes: (i) permitir validar a cadeia de certificados presentes no cartão de cidadão, bem como verificar a elegibilidade de um cidadão através da utilização de uma rede descentralizada de oráculos; (ii) estender o suporte do protocolo para vários candidatos eleitorais; (iii) estender o suporte da Dapp para alternativas ao *Metamask*; (iv) reduzir os custos financeiros e computacionais do sistema, organizando o esforço da execução de diferentes operações lógicas dos contratos inteligentes, em diferentes transações; (v) armazenar as chaves de voto dos eleitores na memória interna do cartão de cidadão eletrónico (não é possível, atualmente, memorizar as chaves de voto de forma segura, devido às restrições impostas às funções de criptografia utilizadas pelo cartão).

Abstract

An electronic voting system that fully mimics real-world systems has long been desired. Until recently, it had not been possible to fully address the mandatory properties of a real-world voting scheme, simultaneously. Recently, with the onset of new technologies and research, however, it is not only possible to fulfill these very properties, but also to improve the anonymity and convenience of voting.

A decentralized and self-tallying electronic voting protocol that substantially enhances the privacy of voters and diminishes centralization is developed in this work and presented in this dissertation. These properties are accomplished through a symbiotic relationship between the Ethereum Blockchain and the Portuguese electronic ID. Unlike previously proposed Blockchain e-voting protocols, this is the first implementation that more closely fulfills most of the security requirements of a real-world voting scheme. Furthermore, this system improves currently in-use e-Voting systems by using a self-tallying protocol. Thus, each voting citizen is able to compute the tally of the election and has complete control over their own vote. The execution of this protocol is enforced using the consensus mechanism that safeguards the Ethereum Blockchain.

To prove its feasibility, its implementation was tested on the official Proof of Work (PoW) test network of Ethereum (known as Ropsten). The financial and computational breakdowns are on par with the leading Blockchain e-voting protocol.

Keywords

Blockchain, e-Voting Protocols, Ethereum, DApp, Decentralized Applications, Smart Contracts

Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface

Dapp Decentralized Application

DPoS Delegated Proof of Stake

E2E End-to-End

eID Electronic ID

EVM Ethereum Virtual Machine

HTTP Hypertext Transfer Protocol

JDK Java Development Kit

JSON JavaScript Object Notation

P2P Peer to Peer

PIN Personal Identification Number

PoS Proof of Stake

REST Representational State Transfer

RSA Rivest Shamir Adleman

SHA256 Secure Hash Algorithm 256

SHA256withRSA Secure Hash Algorithm 256 with Rivest Shamir Adleman

SDK Software Development Kit

SSL Secure Sockets Layer

ZKP Zero Knowledge Proof

Chapter 1

Introduction

This document describes the work performed in the scope of a project for the attainment of a master's degree in Computer Science and Engineering at Universidade da Beira Interior. This dissertation addresses the subject of electronic voting, and how a decentralized application that synergises with an electronic ID may simplify and enhance the processes relating to voting. The ensuing section presents the motivation and scope of the work. Section 1.2 enounces the problem statement and defines the objectives. Section 1.3 discusses the adopted approach. Section 1.4 presents the main contributions of this work. Lastly, section 1.5 describes the overall organization of the document.

1.1 Motivation and Scope

Electoral systems are comprised by a set of rules that determine how elections should be conducted and how their results are decided. These sets of rules encapsulate all aspects of the voting process, such as when an election is to take place, who can vote, who can stand as a candidate, how the ballots are to be labeled and cast, how the ballots are to be counted, among others. To illustrate as an example, in a traditional voting system, a person of age would attend their parish or consular district, provide their identification card to a polling agent and receive a ballot paper with a check box list of the possible candidates to vote for. Following this, the person in question proceeds to an isolated area where they may select a single candidate — or none — of the list with a check mark. The ballot is then delivered to the ballot box, to be counted by a teller once the deadline for voting finishes. After all the votes are counted, by the many tellers across the country, the results are calculated and made available to the public by the government.

An electronic voting system that mimics real-world systems has long been desired. Up until this point, it had not been possible to fully address all the mandatory properties of a real-world voting scheme simultaneously. These properties are, among others, security, anonymity, coercion resistance and untraceability. Recently, with the onset of new technologies and research, however, it is not only possible to fulfill these very properties, but also to improve the anonymity and convenience of voting. One embodiment of such technology is the Ethereum Blockchain. It possesses the following properties:

- It is *practically immutable*, as altering information would require copious amounts of computing power;
- It is *transparent*, as anyone can view its data;

- It is *decentralized*, as there is no central authority behind it;
- It is used to build smart contracts and Dapps.

This Masters project aspires to provide an elegant solution to electronic voting that comprehends the strengths of an electronic ID, coupled with the potential of the blockchain. This allows for a system that deviates, as much as possible, from centralization, the true downfall of any democratic system. Its scope falls in the areas of computer security, digital transformation and system design. Under the 2012 version of the ACM Computing Classification System, a *de facto* standard for computer science, the scope of this Masters project could be placed into the following categories:

- **Applied Computing**∼**Voting / election technologies**
- *Security and privacy*∼*Usability in security and privacy*
- *Privacy protections*
- Centralization / decentralization

1.2 Problem Statement and Objectives

The objective of this Masters project is to study, design and implement an electronic voting system using a Decentralized Application built on top of the Ethereum Blockchain. This master's project may be divided into three main objectives:

1. Study the security challenges associated with electronic voting systems; identify requirements (namely security requirements);
2. Engineer an electronic voting system; implement a set of Ethereum smart contracts and decentralized apps underlying the particular case studied in the scope of this project;
3. Implement a web application that makes use of the Portuguese Citizen Card eID, for authentication purposes, that will allow the user to cast a vote, without knowledge of the underlying system.

Ultimately, the main goal is to transform and migrate a physical voting system used by humans to the digital world and improve it. Current electronic voting systems, despite emulating most of the physical system requirements and functionalities, contain many issues and vulnerabilities. This results in a system that does not fully mimic a traditional voting system. It is our objective to propose and create an electronic voting system that fulfills all the system and security requirements of a physical voting system, and, in addition, enhance and reform the current standard.

1.3 Adopted Approach for Solving the Problem

From a macro perspective, this work followed the typical approach for proposing new cryptosystems: (i) define system and security requirements; (ii) propose a construction that potentially addresses those requirements; (iii) prove that the requirements were met. When applied to cryptosystems, phase (iii) is typically achieved by proving that breaking the system is the same of breaking an intractable mathematical problem. This work focused mostly in phases (i) and (ii), though the next to last chapter addresses phase (iii). These phases can be further divided as follows:

- Study the current physical voting systems, in furtherance of deeply understanding the manner by which they function;
- Describe the system requirements from the point of view of the requirements that an electronic voting system must have;
- Investigate and scrutinize the technologies mentioned below, necessary for the implementation of this system;
- Design the system, encompassing all the system requirements;
- Test the system;
- Prove the fulfillment of the security properties; and
- Analyze the financial cost of the system.

The system was developed through the synergistic usage of the following technologies:

- **Ethereum Blockchain** — necessary for the decentralized execution of smart contracts;
- **Zero-Knowledge Proofs** — required for the voting protocol. Allows voters to not only check that their vote has been recorded and cast as intended, but also to independently compute the tally;
- **Ethereum Smart Contracts** — to contain and execute the logic responsible for standardizing the cryptographic functions used among all the voters, control the electoral process and verify the zero-knowledge proofs;
- **web3.js** — contains a collection of modules with specific functionality to interact with the Ethereum ecosystem, namely the Ethereum Blockchain and smart contracts;
- **React.JS** — to create a single-page web application that interfaces with an eID and interacts with smart contracts through the usage of `web3.js`;
- **Portuguese Electronic ID** — enables personal identification and authentication of a citizen;
- **Node.JS Server** — to perform the personal identity verification of the citizen and ascertain the authenticity of the certificate chain present in the eID;

- **sockets.io** — used for real-time bidirectional event-based communication between the Dapp and the Node.JS server;
- **Java client** — to retrieve the information stored in the citizen card, as well as authenticate a voter using their Personal Identification Number (PIN) number;
- **Loopback Application Programming Interface (API) Server** — necessary for the Dapp to indirectly interface with the eID;

1.4 Main Contributions

The main contributions achieved from the research and development of the idea herein presented can be enumerated as follows:

1. Identification of the security requirements of an electronic voting system, along with the design of a system that fulfills these security requirements employing the ensuing technologies. The security requirements are described in chapter 5 of this dissertation;
2. The design, assembly and testing of a Decentralized Application (Dapp) that interacts with Ethereum smart contracts, concerning to an electronic voting system. Up to this point, there is not a system that integrates these technologies, specifically towards an electronic voting system using an electronic ID.
   2.1. Election officials can completely customize an election smart contract, deciding, for instance, the topic at hand, the deadlines for each of its phases and an optional safety deposit fee. The administrators are able to control the transitions through each of the phases of the election in the Dapp.
   2.2. Voters are able to authenticate themselves using their eID, register for an upcoming election, cast a vote and compute the final tally (once the voting phase has come to an end);

   The system requirements and its design are outlined in the third chapter of this document, whereas the implementation of the Dapp is described in chapter 4;
3. The design, assembly and testing of a communication bridge between the Dapp and the middleware of the Portuguese Citizen Card. This allows the Dapp to request information from the eID, as well as request the voter to corroborate their identity, through a digital signature. The implementation of this system is described in chapter 4;
4. The design, assembly and testing of a server that verifies the validity of the certificates present in the Electronic ID, the authentication of the elector and their voting eligibility. The implementation of this system is illustrated in chapter 4.
5. The verification of the fulfillment of the security requirements, the testing of the system and its financial breakdown. This is described in chapter 5;

6. The developed system was subject to a paper entitled *Blockchain-based Decentralized Application for Electronic Voting Using an Electronic ID*, presented and accepted for publication in the proceedings of the 11º Simpósio de Informática (INForum 2019), held at Universidade do Minho, between the 5th and 6th of September, 2019 [1].

1.5 Dissertation Organization

The organization of this dissertation follows the overall adopted approach for solving the problem, with some minor adaptations, and its contents can be summarized as listed below:

- Chapter 1 — **Introduction** — contextualizes the problem addressed by this Masters project by presenting its motivation and scope, the problem it attempts to solve, the selected approach for solving the conferred problem, the main contributions of this work, and the organization of the document;
- Chapter 2 — **Background and Related Works** — discusses the concepts that served as a cornerstone to this Masters project, as well as related works that have influenced and contributed to it;
- Chapter 3 — **System Requirements and Design** — outlines the requirements for an electronic voting system, and magnifies them to the particular case of smart contracts. Finally, a system that takes into consideration the aforementioned requirements is proposed;
- Chapter 4 — **System Implementation** — discusses the implementation of the different modules of this e-Voting system and their interactions;
- Chapter 5 — **Fulfillment of the Security Requirements, Testing and Financial Breakdown** — debates in which way each of the security requirements was conformed in the developed system. Furthermore, the results of conducted user tests and a financial evaluation of the system is presented;
- Chapter 6 — **Conclusions and Future Work** — presents the final remarks of this work, what was achieved, and future work that can be developed from it;
- Appendix A — **Code Excerpts** — presents some of the lengthier code excerpts as a complement to the discussion of certain processes described in chapter 4.

Chapter 2

Background and Related Works

2.1 Introduction

This chapter will describe different topics that provide the foundations for this Masters project. To begin with, a brief explanation of smart contracts and their relationship with Ethereum, as well as a thought experiment concerning Zero Knowledge Proofs (ZKPs), in a section named **Background** (section 2.2). In addition, the current state of the art regarding electronic voting, as well as its dilemmas are discussed in the intermediate section 2.3.

2.2 Background

Four topics were extensively researched, which served as a cornerstone to this Masters project: (i) Ethereum and decentralized applications; (ii) smart contracts; (iii) electronic voting; and (iv), zero-knowledge proofs.

2.2.1 Ethereum and Decentralized Applications

Ethereum is a distributed computing platform and operating system, initially described in December of 2013 by Vitalik Buterin [6]. The original goal of Ethereum was to provide a platform in which decentralized applications could be built and run on.

The Ethereum blockchain is secure by design. It is *practically immutable*, as altering information would require extensive amounts of computing power. It is *transparent*, as it can be observed by anyone. It is *decentralized*, as there is no central entity governing it.

A decentralized application, or Dapp, is an application that runs on a Peer to Peer (P2P) network of computers, rather than on a single computer. Traditionally, a web application renders data to the front-end, obtained from a centralized database or API. A Dapp, however, obtains its data from a smart contract, provided by a decentralized P2P network. As such, the organization that created the Dapp is not able to manipulate the data stored in the network, granting it immutability and revoking any rights to central authority.

2.2.2 Smart Contracts

Smart contracts are Turing-complete programs that run in a decentralized manner. They do not require a central authority to verify their fulfillment. All the parties that agree to the logic presented in the algorithm of the smart contract become, therefore, bound to it. This means that any problem that requires an overseeing and dictatorial authority is able to be solved, in a trust-less fashion. Furthermore, the code of each and every smart contract is public, implicating that anyone can verify its logic. Smart contracts are also immutable. As such, the logic represented in them can never be changed after their deployment to the network. Most importantly, all the peers in the underlying P2P network independently execute the contract code, which allows consensus to be reached on its output.

In this Masters project, the choice of using Ethereum smart contracts was made. Ethereum supports smart contracts written in seven different languages. These are compiled to Ethereum Virtual Machine (EVM) bytecode and deployed to the Ethereum Blockchain for execution. The EVM is the runtime environment of Ethereum. It is designed to run smart contracts exactly the same way, each and every time, across an ever-growing network of international public nodes. It is its intrinsic consensus mechanism that allows for decentralization in Ethereum. An excerpt of the smart contract code used in this project is presented in listing A.5, included in appendix A.

2.2.3 Electronic Voting

Electronic Voting, or e-voting, as will be referred to onward, is an alternative system of casting and counting votes. It brings great advantages to an increasingly globalized and digital society. E-voting is cheaper, increases voter turnout and confidence, and transforms a process that has existed for thousands of years in one that is exceptionally efficient and on par with cultural and technological advancements [7]. There are still, however, many caveats to this process, predominantly concerning security. E-voting systems, in particular the one used in Estonia, have been thoroughly criticized by [8], the greatest root for concerns being its inherent centralization.

> “Those who vote decide nothing. Those who count the vote decide everything.” - Joseph Stalin

Ultimately, to protect a democratic electoral system from corruption, centralization must be removed.

I-Voting is a specific embodiment of e-voting, in which voting is conducted via the Internet. Consequently, a voter is not required to travel to a polling station in order to cast a ballot, being able to do so from the comfort of their home. As it will be further discussed in this chapter, Estonia allows its citizens to vote through the Internet.

2.2.4 Zero-knowledge Proofs

Zero-knowledge proofs were first devised in 1985 by Shafi Goldwasser, *et al*. [9]. In lay terms, a zero-knowledge proof allows for a subject A to prove to a subject B that he is in possession of certain information, without revealing the information itself. To demonstrate, let us consider the following experiment.

Let us imagine that a party, Alice, is red-green colour-blind, while another party, Bob, is not. Bob possesses two completely identical spheres, except for the fact that one of them is red and the other one is green. Bob wants to prove to Alice that they have, in fact, different colors, omitting any other information (namely their exact colors). In fact, Bob does not want to reveal which sphere is red and which one is green. The proof system is as follows:

1. Bob gives both of the spheres to Alice, and she puts them behind her back;
2. Alice chooses one of the balls and displays it;
3. Alice, once more, places the displayed sphere behind her back, shuffles them, and reveals one of the two;
4. Alice asks Bob if the sphere is the same or if it has changed.

This procedure can be repeated *ad nauseum*. If the spheres were to be the same color, Bob would not be able to guess correctly with probability higher than 50%.

There are certain properties that a ZKP must adhere to:

- Completeness – If a statement is true, an honest prover (i.e., one that is following the protocol correctly) will be able to convince an honest verifier of that fact. In the thought experiment above, it corresponds to the fact that the spheres are indeed differently colored, as stated by Bob, the honest prover;
- Soundness – If a statement is false, it is probabilistically very unlikely that a cheating prover can convince an honest verifier otherwise. In the example above, it is ascribed to the fact that Bob would have a nearly zero probability of consistently identifying whether the sphere was switched or not if they were of the same color;
- Zero knowledge – If a statement is true, the only information that the verifier is able to obtain is the fact that the statement is true. In the example above, Bob only ever reveals to Alice if the sphere was switched or not. No other information (namely the colors of the spheres) is revealed.

2.3 Related Works

This section describes an array of different works that served as a stepping stone to the development of this Masters project.

2.3.1 Interesting works from Specialized Literature

This subsection discusses some of the works that may be considered closely related with this one, presenting some of their interesting points.

2.3.1.1 Self-tallying Elections and Perfect Ballot Secrecy

Kiayias and Yung, in 2002, introduced a new election paradigm [10]. This voting system assures three different properties: Perfect Ballot Secrecy, Self-tallying and Dispute-freeness. Previous schemes that supported perfect ballot secrecy lacked fault tolerance. This means that a faulty participant would be able to impair the election. Furthermore, they also required voter-to-voter interactions, whereas the aforementioned voting scheme does not. It avoids this by employing dispute-freeness and publicly verifiable messages, and suggesting an innovative *corrective fault tolerant* mechanism, which neutralizes faults from occurring before and after a ballot is cast. As previously mentioned, this protocol adopts self-tallying. Consequently, it allows any voter or third-party observer to compute the tally once all ballots are cast. This removes an additional layer of centralization from the election, as a tallying authority is no longer required.

2.3.1.2 Anonymous voting by two-round public discussion

Hao *et al.* [2], proposed large improvements to the Kiayias-Yung protocol [10], by reducing the number of rounds to two while, concurrently, reducing the number of exponentiations effected to one per voter. The Kiayias-Yung [10] protocol had previously had its system complexity improved by Groth [11], however, its performance increases were not optimal. This was due to the fact that their approach relied on a trade-off between round efficiency and less computation. The more recent proposal is, however, significantly more efficient in terms of the number of rounds, computational cost, and bandwidth usage. It was first implemented by the Open Vote Network [4] (see section 2.3.4).

2.3.1.3 Towards Secure e-Voting Using Ethereum Blockchain

Emre Yavuz *et al.* [12], devise and implement an election voting system using Ethereum smart contracts. In the developed smart contract, they consider two entities: a *chairPerson* and a *voter*. The *chairPerson* has the power to entitle an Ethereum address as *voter*, allowing said address to vote. Nonetheless, the results of an election can be queried at any time, by anyone, which gives an unfair advantage to those who have yet to vote. Furthermore, this system does not comply with the privacy of voters, integrity, verification and non-repudiation of votes, and transparency of counting.

2.3.2 Follow My Vote

Follow My Vote [3] provides a Delegated Proof of Stake (DPoS) solution to online voting. Accordingly, the voter is able to delegate their influence in the network to a delegate that they consider trustworthy. In a Proof of Stake (PoS) system, the influence of a user in the network is based on their own affluence or maturity. The protocol behind Follow My Vote requires an initial registration phase, in which the credentials of a user (picture and government ID card) are verified by a centralized entity. While Follow My Vote claims to use blind signatures to provide anonymity after the vote is submitted to the Blockchain, the registration phase inherently provides a worrisome vulnerability to the protocol, since the voters are onymous to the central authority responsible for the verification.

2.3.3 TIVI

TIVI [13], much like Follow My Vote [3], only uses the Blockchain as a ballot box that stores data related to the voting process. It innovates by using a cryptographic *mixing* process, which decouples the identifying information of a voter from their encrypted vote. Nonetheless, the protocol requires an external authority to verify the eligibility of a voter. Alongside with Follow My Vote, the need for a central verifying authority is a liability to the protocol.

2.3.4 Open Vote Network

Patrick McCorry *et. al.*, present the first implementation of a decentralized and self-tallying internet voting protocol [4]. This system is able to retain maximum voting privacy by using the Ethereum Blockchain. Initially crafted by Kiayias and Yung [10], and later improved by Hao *et al.*[2], this protocol pioneered the needlessness for a trusted authority in the processes of calculating the tally and protecting the anonymity of a voter. The Open Vote Network is a self-tallying protocol in which the privacy of a vote is controlled by its corresponding voter. Thus it follows that only a full collusion amongst all the voters would allow for a single vote to be exposed. The execution of this protocol is enforced by the same consensus mechanism that safeguards the Ethereum Blockchain. Their Schnorr non-interactive ZKP and 1 out of 2 ZKP functions to create and verify ZKPs on an Ethereum smart contract have been adopted by this work.

2.3.5 Estonian Internet Voting

Estonia has been a proponent of Internet voting since 2005, when it first held municipal elections using it [14]. Shortly after, in 2007, it became the first country in the world to employ internet voting in parliamentary elections. Nowadays, more than 30% of its ballots are cast online, according to Tarvi Martens, Chairman of the Estonian Electronic Voting Committee. Nonetheless,

their i-voting system (the i-voting concept refers to e-voting supported specifically by the Internet – see section 2.2.3 for a better description of these concepts) has been thoroughly criticized due to its serious architectural limitations and procedural gaps [8]. Furthermore, its lack of End-to-End (E2E) auditability is worrisome, as it has to rely on the integrity of the computer of the voter, server components and election officials. Cryptographic techniques that achieve end-to-end auditability enable individual voters to verify that every vote has been counted accurately. In order to provide transparency to the system, live video recordings were provided to the general public. Yet, in some instances, workers unintentionally typed root passwords for the election servers in view of the camera and used personal flash drives to move the official election results off of the counting server, as emphasized in [8]. Moreover, the election staff was recorded using personal computers to prepare the client software for the election, which would later be distributed to the public, creating the possibility of spreading malicious code to the system of the voter.

2.3.6 Comparison Between the Existing i-Voting Protocols

Most of the aforementioned e-voting systems are subject to a comparison regarding the fulfillment, or not, of e-voting requirements, according to [5], in table 2.1. These requirements are further clarified in section 3.2. The technologies employed, which promote these very requirements, are also subject to an analysis in table 2.2.

Requirements / e-Voting Systems	Accuracy	Privacy	I&UV*	Eligibility	Fairness	Decentralized
Our Project	✓	✓	✓	✓	✓	✓
Open Vote Network	✓	×	✓	✓	✓	✓
TIVI	✓	×	✓	✓	✓	✓†
Follow My Vote	×	×	✓	✓	✓	✓†
Towards Secure e-Voting	✓	×	✓	✓	×	✓
Estonian i-voting	×	✓	×	✓	✓‡	×

Table 2.1: Comparison between e-voting systems regarding requirements (* refers to *Individual and Universal Verifiability*, † highlights that the protocol does not use a PoS/PoW Blockchain and ‡ indicates that the requirement may be compromised by a dishonest tallying authority).

As will be discussed in chapter 3, privacy as a requirement to an e-voting system refers to the inability to link a voter to its vote. Protocols that rely on a central authority to verify the eligibility of voters and, proven eligible, attribute them voting keys, compose a vexing vulnerability to the system itself, as this central authority is able to link a voter to its vote. Therefore, there is a direct cause relationship between the privacy requirement being fulfilled and the system using an eID for authentication.

Technologies / e-Voting Systems	E2E Auditability	Authentication Using eID	Immutability	Smart Contracts	Zero-Knowledge
Our Project	✓	✓	✓	✓	✓
Open Vote Network	✓	×	✓	✓	✓
TIVI	✓	×	✓	×	×
Follow My Vote	✓	×	×	×	×
Towards Secure e-Voting	✓	×	✓	✓	×
Estonian i-voting	×	✓	×	×	×

Table 2.2: Comparison between e-voting systems concerning technologies.

2.4 Conclusions

This chapter described some topics that were considered to be important for the complete understanding of this work. Furthermore, different works that were crucial for grasping the state of the art in regard to e-voting systems were depicted, including their advantages and flaws. This made possible the discovery of which technologies fulfilled the various e-voting system requirements. None of the voting systems evaluated conformed with each and every security requirement. The work that more closely fulfill these requirements was the Open Vote Network, which only lacked the **privacy** requirement. Our work intends to implement a system using similar technologies to this one, but with increased privacy through the means of using a government-issued electronic ID for voter authentication.

Chapter 3

System Requirements and Design

3.1 Introduction

This chapter contemplates and presents the different system requirements, along with its design. The e-voting system requirements, including a general and a particular system using smart contracts is defined in section 3.2. Furthermore, the software system requirements are delineated in section 3.3. Finally, a proposal for the system is outlined in section 3.4.

3.2 E-Voting System Requirements

An electoral system is comprised of a set of rules and protocols that regulate all the facets of the voting process. This section will discuss the voting requirements necessary for an e-Voting system, as well as the particular specifications that an e-Voting system using smart contracts must employ.

3.2.1 General e-Voting System Requirements

Electronic Voting systems seek to improve the conventional electoral system, by modernizing it on par to the cultural and technological advancements. In a traditional electoral setting, the voting process follows a very specific process:

1. A list of eligible voters is collected;
2. The eligibility of the voters is checked, ensuring that only legitimate electors can cast their vote;
3. The ballots are collected;
4. The tally is calculated by counting the votes.

This pattern must be closely matched by its electronic counterpart. Furthermore, according to [5], a case study on the design and implementation of an e-voting prototype system, the following core properties must be assured:

- **Accuracy** — It must not be possible to alter a vote;
- **Privacy** — A voter must not be linked to their vote;
- **Individual and universal verifiability** — Any given voter must be able to verify that their vote has been cast correctly, as well as verify that all votes have been counted properly (it may not be possible to entirely fulfill this property in legacy voting systems);
- **Eligibility** — The system must only allow eligible voters to vote; and they should only be able to do so once;
- **Fairness** — The results of an ongoing election must not be disclosed before its end.

Some of these security requirements are refinements of the desired properties of a real-world voting system transposed over to the electronic realm, except for coercion resistance. Coercion resistance can be seen as an extension of the basic property of privacy. Privacy, in the electoral system, is fulfilled when an opponent can not interact with the voters during the election process. Coercion resistance is a strong form of privacy in which the attacker is believed to be able to interact with the voter. In particular, the attacker may require targeted voters to reveal their private keys after registration, or may request that such voters cast a particular form of ballot. If the attacker can ascertain whether or not the electorate acted as directed, they will be able to extort or otherwise exert undue influence over the electoral process. Therefore, a coercion-resistant voting system is one in which the user can trick the attacker into believing that they have acted as directed, when the elector has actually cast a ballot in accordance with their own desires.

Aside from these inherent properties of the democratic process, elucidated above, additional underlying properties are imperative for the ratification of electronic voting:

- **Robustness** — In other words, the election must not be affected by faulty procedures or illegal behaviour;
- **Mobility** — Namely, voters should not be constrained by their geographical location;
- **Integrity** — That is, the computer system responsible must be tamper-proof, data integrity must be kept, and it should be collusion resistant against electoral agencies;
- **Convenience** — Particularly, users should be allowed to cast their vote easily, quickly and with minimal instruction.

In current traditional and e-voting systems, none of the properties above are fulfilled, neither partially nor fully. The reason for this is the centralized nature of the system itself. A system cannot be collusion resistant against the same authorities that regulate it. Current voting systems have the following predicaments:

- The system *is not fair* — Electoral authorities are the only ones allowed to compute the tally and might be aware of the results before they are made available to the general public;

- The system *is not accurate* — Ballots can be easily tampered with [15];
- Eligibility can be overridden — Votes from ineligible voters can be introduced by the electoral authorities [16, 15];
- The integrity of the system is easily compromised through corruption and hacking.

3.2.2 Particular Specifications for an e-Voting System Using Smart Contracts

During the last century, several changes have improved the robustness of the electoral system that governs different democratic societies around the globe [17, 18]. Notwithstanding, its largest dilemma, collusion within the electoral authorities, can only be solved by a complete revamp of the system itself. Traditional electoral systems, electronic or otherwise, reinforce the authority of the state. The process is obscured and centralized. In a smart contract-enabled electoral system, the electoral process is transparent and decentralized. This asserts the authority of the people.

The proposed system intends to largely increase the sturdiness and convenience of an electoral process, by achieving the following properties:

- The system is truly **fair** — Anyone is able to compute the tally, and only allowed to do so at the end of the elective process;
- The system is truly **accurate** — Only a collusion between 51% of the network would allow for votes to be tampered with [6];
- The system is further **mobile** — Anyone is able to cast their vote from wherever they are, provided that they own a computer or smartphone, internet connection and a card reader;
- The **robustness** and **integrity** of the system is largely improved — The correct execution of the protocol is enforced by the same network consensus that secures the Ethereum blockchain;
- The anonymity of the voter is greatly increased — Only a full collusion amidst all the voters would allow for the identity of a single voter to be revealed [4].

3.3 Software System Requirements and Modeling

Software system requirements, depicted in section 3.3.1, describe the features and behavior of the application. The data entities and relationships of the database used are described in a section named **Data Models and Formats** (section 3.3.2).

3.3.1 Requirements Analysis

There are several use cases for the developed voting system, herein represented by the flowcharts in figures 3.1 and 3.2. As portrayed in these illustrations, two different actors exist: the **administrator** and the **voter**. The administrator has the role of managing and executing the election, while the voter has the objective of authenticating themselves, submitting their voting keys, casting a vote and, optionally, computing the tally.

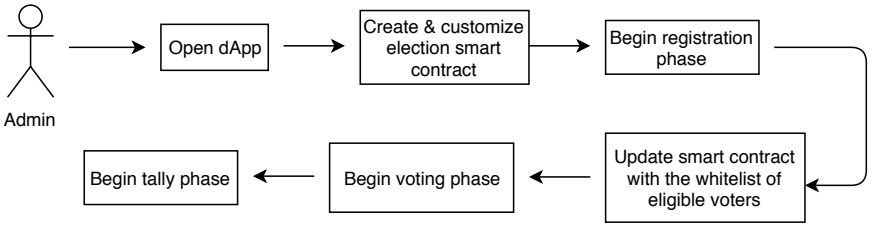

Figure 3.1: Flowchart regarding the administration process of the e-Voting system.

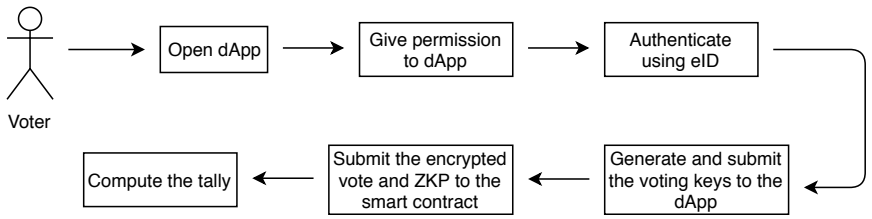

Figure 3.2: Flowchart pertaining to the voting process of the e-Voting system.

The use case related with registering for an upcoming election is described in table 3.1. This process is accomplished effortlessly from the perspective of the voter, since the only interaction between them and the system is the input of the authentication PIN of the eID.

Optimal Scenario	
Precondition	The user has the Metamask extension installed on their browser, with a wallet configured; The PORelD middleware is running, along with the local Java client and the LoopBack API server; There is a card reader connected to the machine, which includes a Portuguese eID connected to it.
Summary	1. The use case begins when the user presses the ”Authenticate” button; 2. The user introduces the PIN of the eID correctly when prompted to do so.
Postcondition	The user is authenticated successfully.

Table 3.1: Optimal scenario for the use case *Authentication of a voter using an eID*.

The administrator must undergo several use cases, from the inception to the end of an election. Such use cases are *the whitelisting of public keys* that will be allowed to interact with the smart contract (described in table 3.2), *the configuration of the different parameters of the election* (illustrated by table 3.3), *the advancement through the different stages of the election* (depicted in tables 3.3, 3.4) and *the computation of the tally* (presented in table 3.5). The use case concerning to the submission of a vote, by a voter, is presented on table 3.6. Regarding the casting of a vote, by a voter, its use case is described on table 3.7.

Optimal Scenario	
Precondition	The administrator has Metamask installed on their browser, with a wallet configured; The public key of the Metamask wallet is the same as that of the owner of the smart contract; The election is currently in its starting phase.
Summary	1. The use case begins when the administrator opens the Dapp; 2. The administrator presses the ”Cloud” button to retrieve the authenticated public keys; 3. The administrator presses the ”Update” button. 4. The administrator presses the ”Confirm” button on the Contract Interaction dialogue.
Postcondition	The administrator updates the smart contract with the public keys allowed to interact with it.

Table 3.2: Optimal scenario for the use case *Whitelisting of public keys by the administrator*.

Optimal Scenario	
Precondition	The administrator has Metamask installed on their browser, with a wallet configured; The public key of the Metamask wallet is the same as that of the owner of the smart contract; The election is currently in its starting phase; There are at least three whitelisted addresses on the smart contract.
Summary	1. The use case begins when the administrator opens the Dapp; 2. The administrator presses the ”Next” button; 3. The administrator selects the desired deadlines for the different stages of the election; 4. The administrator presses the ”Next” button; 5. The administrator inputs the topic of the election; 6. The administrator presses the ”Next” button; 7. The administrator presses the ”Confirm” button on the Contract Interaction dialogue.
Postcondition	The administrator configures the election smart contract and begins the registration phase.

Table 3.3: Optimal scenario for the use case *Configuration of the election by the administrator*.

3.3.2 Data Models and Formats

This project makes use of LoopBack API in order to handle the communications between the Dapp and the local Java client (further details regarding these two applications may be inspected in sections 3.4, 4.2 and 4.3). The LoopBack API server was configured using five different model endpoints, each of which has an associated database table, as displayed in figure 3.3. Table `citizens` contains the different properties relating to the card of a citizen, accordingly, the full name, date of birth, identification number, beginning and end of the validity of the citizen card, authentication certificate and certificate chain. Table `middlewareStatus` holds the *boolean* properties pertaining to the status of the local client, namely, whether the `PTeID` middleware and the local java client are running, the card reader and the citizen card is detected, and the certificates are successfully read. Table `requestData` includes the properties concerning the type of request sent from the Dapp to the client (the different types of requests available are covered in section 4.3.1). Table `requestSignature` contains the properties relating to the token that will be signed by the client. Table `signature` includes the properties pertaining to the signature created using the token provided by the Dapp.

Optimal Scenario	
Precondition	The administrator has Metamask installed on their browser, with a wallet configured; The public key of the Metamask wallet is the same as that of the owner of the smart contract; The election is currently in its registration phase; The deadline for the registration has elapsed.
Summary	1. The use case begins when the administrator opens the Dapp; 2. The administrator presses the “Next” button; 3. The administrator presses the “Confirm” button on the Contract Interaction dialogue.
Postcondition	The administrator advances to the voting phase of the election.

Table 3.4: Optimal scenario for the use case *Advancement to the voting phase of the election by the administrator*.

Optimal Scenario	
Precondition	The user has Metamask installed on their browser, with a wallet configured; The election is currently in its voting phase; The deadline for the voting phase has elapsed.
Summary	1. The use case begins when the user opens the Dapp; 2. The user presses the ”Calculate Tally” button; 3. The user presses the ”Confirm” button on the Contract Interaction dialogue.
Postcondition	The user advances to the final phase of the election, and the results are displayed.

Table 3.5: Optimal scenario for the use case *Computation of the tally*.

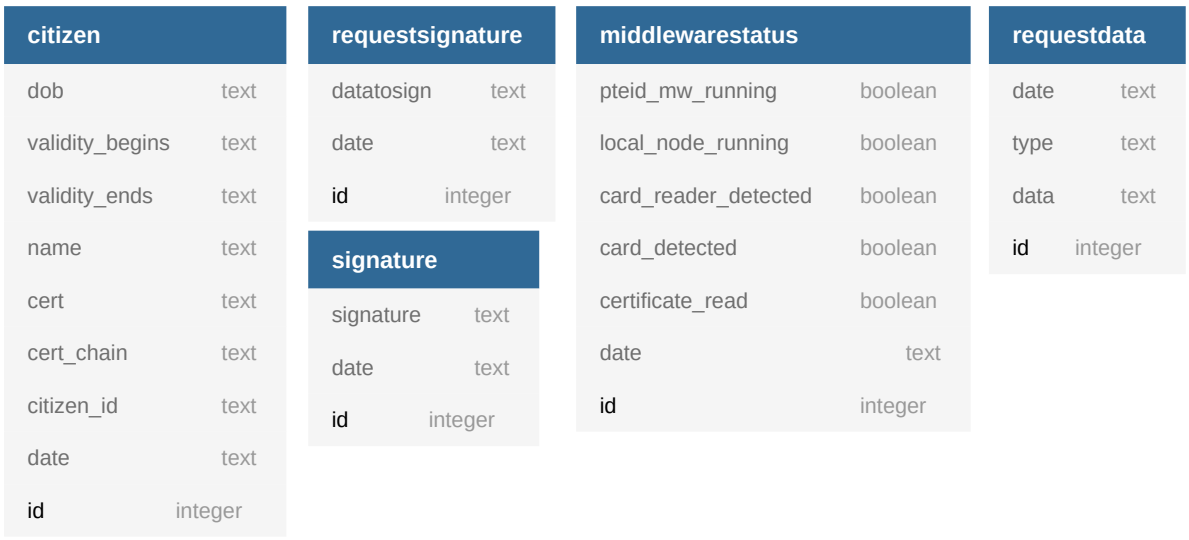

Figure 3.3: Relational model of the database used by the LoopBack API for the communication between the Dapp and the local Java client.

3.4 Proposal for the System

The proposed system is comprised of an array of interconnected programs whose architecture is schematized in figure 3.4. Foremost, the central piece of the system is the Web App, the only part of the system that the user directly interacts with. The Web App can communicate directly with two servers: (i) a remote server running Node.JS and (ii) a local server running

Optimal Scenario	
Precondition	The voter has the Metamask extension installed on their browser, with a wallet configured; The public key of the Metamask wallet of the voter has been whitelisted by the administrator of the election; The election is currently in its signup phase.
Summary	1. The use case begins when the voter opens the Dapp; 2. The voter generates their voting keys independently; 3. The voter uploads their voting keys via drag-and-drop; 4. The voter presses the “Next” button; 5. The voter presses the “Confirm” button on the Contract Interaction dialogue.
Postcondition	The voter registers their voting keys successfully.

Table 3.6: Optimal scenario for the use case *Registration of the voting keys by a voter*.

Optimal Scenario	
Precondition	The voter has the Metamask extension installed on their browser, with a wallet configured; The public key of the Metamask wallet of the voter has been whitelisted by the administrator of the election; The election is currently in its voting phase.
Summary	1. The use case begins when the voter opens the Dapp; 2. The voter uploads their voting keys via drag-and-drop; 3. The voter presses the “Next” button; 4. The voter presses the ”Yes” or the “No” button; 5. The voter presses the “Confirm” button on the Contract Interaction dialogue.
Postcondition	The voter casts their vote successfully.

Table 3.7: Optimal scenario for the use case *Casting of a vote by a voter*.

LoopBack. The objective of the former is to validate the credentials of the voter (such as digital signatures and associated certificates), as well as verify their eligibility for voting. The latter serves as a relaying tool for the communications between the Web App and a local Java Client. This particular Java client runs a library of the Portuguese eID middleware, which means that it is able to interact with the citizen card of the voter. Through this interconnected system, it is possible for the Web App to indirectly interface with the citizen card and, concurrently, verify its eligibility. The authentication segment of the system still bears some limitations in regards to centralization (the verification server used for validating the certificate-chain could be subject to corruption or hacking), which could be solved by using a decentralized network of oracles, as described in a section named **Future Work** (section 6.2).

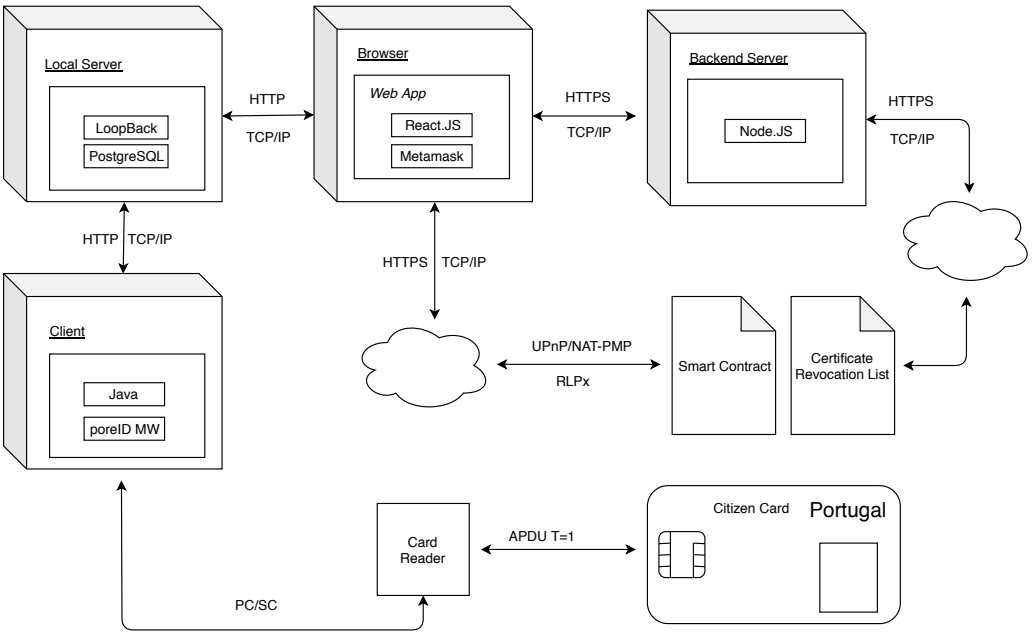

Figure 3.4: Architecture for the e-voting system proposed in the scope of this work with specifications of the main components, technologies and their communication channels.

As mentioned in section 2.2.1, the developed Web App should rather be classified as a Dapp, as it is able to communicate with Ethereum smart contracts. It does so by using Metamask, a browser extension that works as an advanced Ethereum wallet, able to not only send funds to other wallets, but also to interact with smart contract functions. The voting protocol used is based on the Open Vote Network two-round protocol, discussed in section 2.3.4, which uses zero knowledge proofs as a way to allow voters to not only check that their vote has been recorded and cast as intended, but also to independently compute the tally, in order to verify that cast votes have been tallied as recorded. This voting protocol will be established over two different smart contracts. The first, titled *cryptography contract*, is responsible for creating the two types of zero knowledge proofs that standardize the cryptography used among all the voters. The second, titled *voting contract*, controls the election process and verifies the zero knowledge proofs. A brief explanation of zero knowledge proofs is given in section 2.2.4.

Fundamentally, the main role of the Dapp is to encompass all the voting activities into a single platform in order for the voting process to be as simple as possible for voters and election officials alike. These activities are as follows:

- Creation of a customized election smart contract by the administrator;
- Voter registration for an upcoming election;
- Voter authentication using a government-issued eID;
- High-level interaction with different smart contract functions;
- Calculation of the final tally and verification of the well-formedness of the encrypted votes.

3.5 Conclusions

In order to successfully plan the design of this system, requirements relating to e-voting had to be studied, specified and explained. These requirements pertained to general e-voting system requirements, which are refinements of the desired properties of a real-world voting system rendered to the electronic domain, as well as additional properties that are necessary for the reliability of electronic voting. Furthermore, an analysis over the different software requirements was accomplished through the creation of a pair of flowcharts, which revealed several use cases for the different types of users of the system. With this knowledge, it was possible to conceptualize a proposal for the system as a whole.

Chapter 4

System Implementation

4.1 Introduction

This chapter will cover the implementation details concerning the different components of the e-Voting system and their interactions. This includes the approach taken to develop the decentralized web application in section 4.2, the authentication system in section 4.3 and the Ethereum Smart Contracts in section 4.4.

4.2 Implementation of the e-Voting Dapp

As described in section 3.4, the Dapp is the most pivotal part of the system. The reason for this is that the Dapp holds the logic that governs and supervises the remaining elements of the system. It interfaces with the eID of the voter through a LoopBack API server and a local Java client, which allows it to retrieve the information stored in the citizen card, as well as request the voter to sign the authentication token using their PIN. An example of a generic POST request sent from the Dapp to one of the endpoints of the LoopBack API server is presented in listing A.3, included in appendix A. All of the data sent from the client, through the LoopBack server, to the Dapp is, thereafter, validated by a Node.JS server. These validations include the authenticity of the certificate chain and the personal identity verification of the citizen. These processes are covered in more detail in sections 4.2.1 and 4.3. Throughout this work, the Web App will be referred to as Dapp due to the fact that it is, actually, a decentralized web application. Accordingly, the data and the methods related with the voting scheme are stored, not in a trivial database, but rather in a smart contract that is stored and executed by a decentralized blockchain. Further details regarding the Ethereum Blockchain, decentralized applications and smart contracts may be found in section 2.2.1 and 2.2.2.

There are two distinct workflows provided by the Dapp. They diverge according to the requirements of the two types of users in this system, the `administrator` and the `voter`. On the one hand, the `administrator` workflow, represented in figure 3.1, requires an interface that simplifies the management of the election. On the other, the `voter` workflow, expressed in figure 3.2, requires an interface that allows for a quick and straightforward on-boarding, voting and analysis of the final results. Both the administrator and the voter are identified by the public address of the account that they are using on Metamask. The following sub-sections will discuss, in detail, the aforementioned user interfaces and how they abstract the rather complex logic associated with the system from its users.

4.2.1 Administration Workflow

The administrator of the system features a rather different user interface comparing to a voter. It is in this interface that all of the operations regarding the management and execution of the election are accomplished. Before an election is able to be started, the administrator must set up its different configurations. These configurations correspond to the whitelisting of the public keys of each voter, the selection of the deadlines for each phase, the topic of the election and an optional deposit fee for each registering voter. The whitelisting of the public keys, represented on figure 4.1, allow the administrator to query the public keys of each successfully authenticated voter by pressing the `Cloud Download` button. Furthermore, the administrator can update the election smart contract to preserve these addresses by pressing the `Update` button. Only the whitelisted addresses will be allowed to interact with the Dapp and, to the same extent, the election itself.

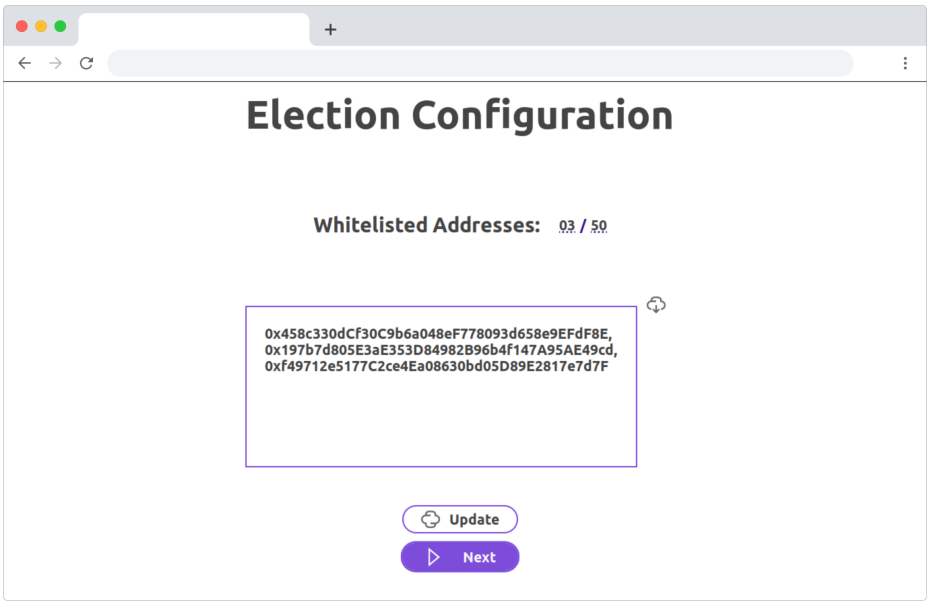

Figure 4.1: Whitelisting of the public addresses.

After pressing the `Next` button, the administrator is taken to the deadline selection view, represented by figure 4.2. In this view, the deadlines of the registration, voting and refunds are chosen by picking their calendar date and time. Due to each of these dates corresponding to a different stage of the election, the period of an earlier phase must not supersede that of a later phase. For instance, the deadline selected for the end of voting must not be earlier than the registration, or later than the end of the refund period. The verification regarding the correctness of the selected dates is done, not only by a simple conditional statement, but also by attempting to locally call the smart contract function responsible for setting the dates. Locally calling a smart contract function does not expend any gas (the term gas refers to the amount of computational effort necessary to execute certain operations in the EVM), since there is no need to interact with the network, but allows the observation of its expected behaviour. Only when the selected deadlines are ascertained to be correct, may the administrator press the `Next` button, and move on to the third, and last, configuration view.

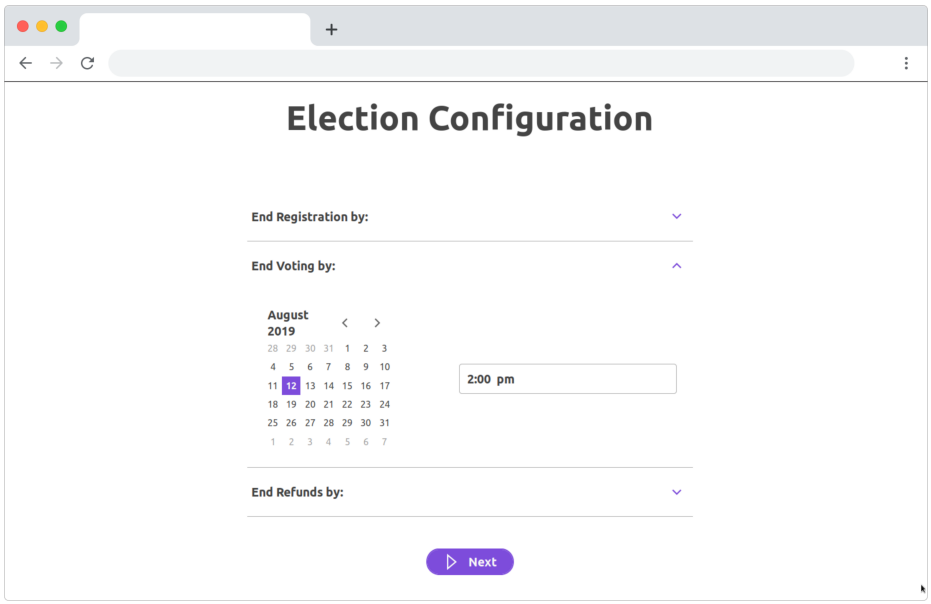

Figure 4.2: Configuration of the deadlines of the election.

The final configuration page, depicted in figure 4.3, allows the administrator to select the topic of the election and, optionally, a safety deposit amount. This value corresponds to the quantity that must be staked by a voter, in `Ether`, when registering to the election. Peculiarly, this value must also be converted, behind-the-scenes, to `Wei`, the smallest unit of Ether (1 `Ether` corresponds to 1×10^{18} `Wei`). Once the administrator presses the `Next` button, a Metamask window responsible for the contract interaction will pop-up. By pressing confirm, the transaction is sent to the network, the smart contract is updated with these configurations, and the election advances to its registration phase.

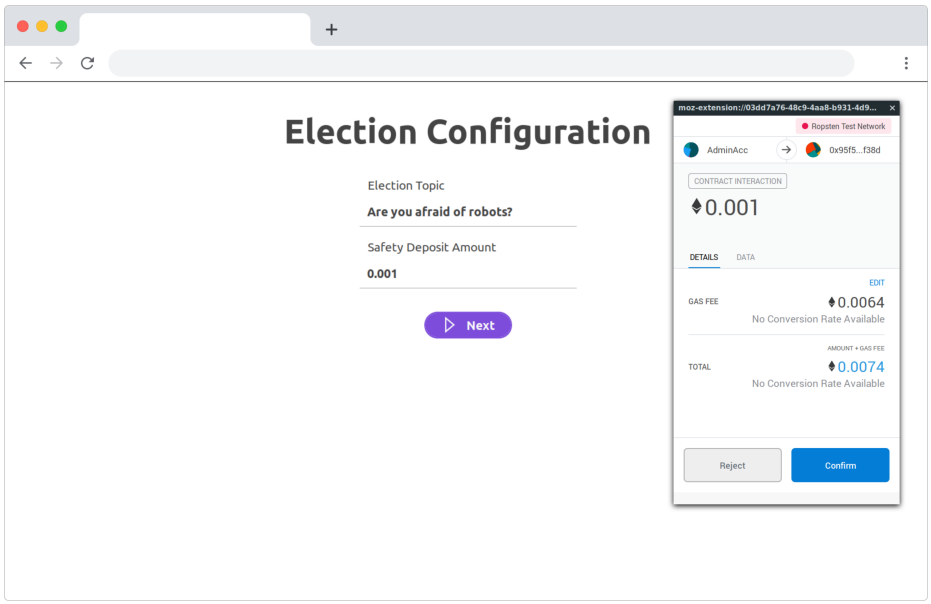

Figure 4.3: Configuration of the election topic and deposit fee.

During the registration phase, the administrator is able to follow the progress of the registration

of voters, portrayed by figure 4.4. As will be described in the next subsection, the registration of a voter corresponds to the submission of their voting keys to the smart contract. Once enough time has elapsed, a `Next` button appears, and the administrator is able to proceed to the voting phase by pressing that button, which triggers a contract interaction transaction. This interaction, which will be described in section 4.4.2, reconstructs the voting keys submitted by the voters and stores them in the voting smart contract.

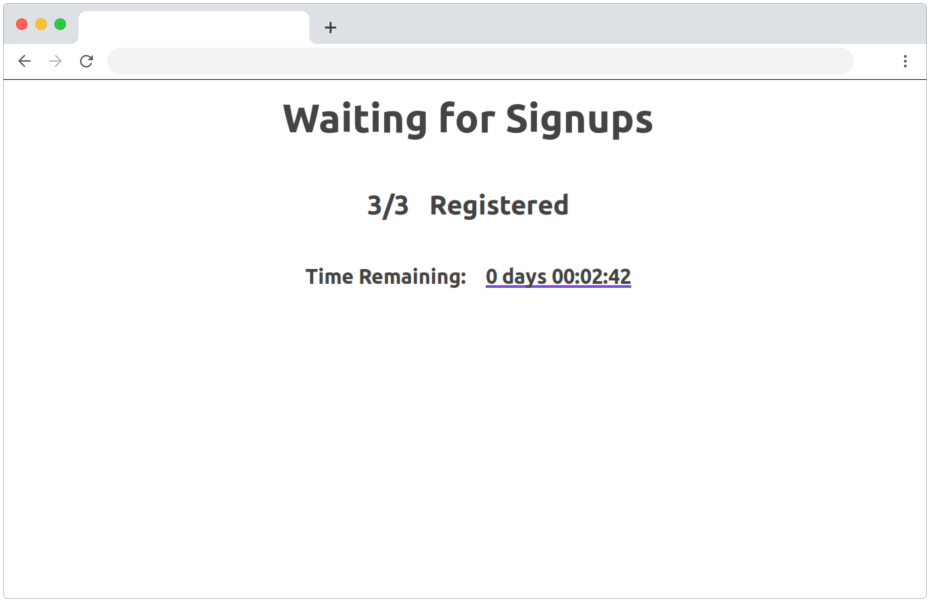

Figure 4.4: Administrator or voter awaiting for the registration deadline.

During the voting phase, in the same fashion as the registration phase, the administrator is able to follow how many users have voted and, once enough time has elapsed, press the `Calculate Tally` button. This instructs the smart contract to calculate the tally and move the election onto its last phase. The final stage of the election produces a view where the administrator and the voters may view the final results. This is depicted in figure 4.5. Pressing the `End Election` button causes a contract interaction transaction, which resets the smart contract to its default values, effectively rebooting the election to its beginning phase.

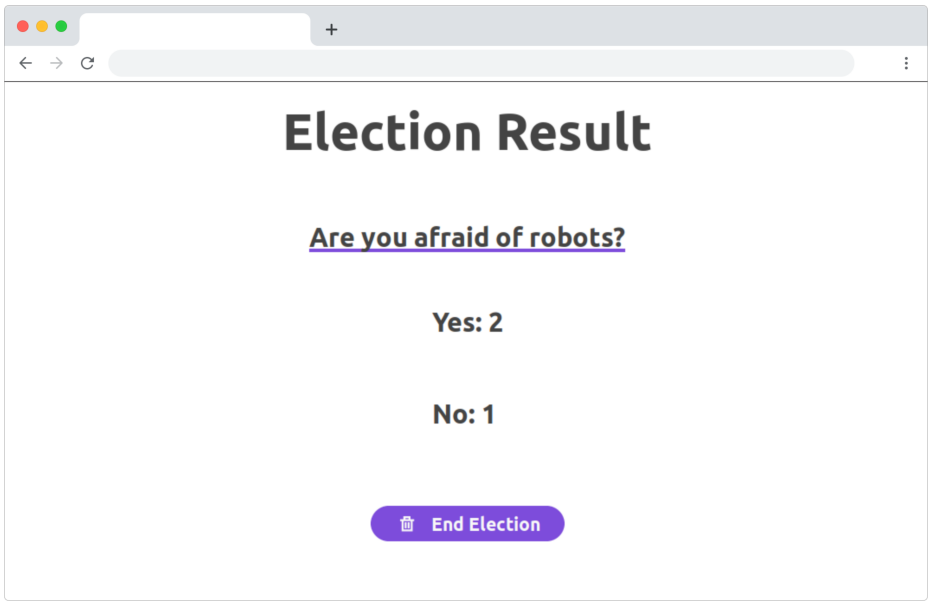

Figure 4.5: View of the results of the election.

4.2.2 Voting Workflow

When a user opens the Dapp, several verifications ensue, the purpose of which are to check that the client has everything required for the authentication mechanism. If a user does not meet some of these requirements, a notice will be raised to inform of the problems, and the user will not be allowed to proceed until each and every one of them are fulfilled. The requirements are the following:

- The `Metamask` extension is installed and running, and an account is logged in. If an account is not currently logged in, but the extension is running, the user will be prompted to do so;
- The local `Java` client is running, as well as the `LoopBack API` server responsible for its communication with the Web App;
- The `PTeID middleware` is running;
- A card reader is connected to the computer;
- A valid eID is plugged to the card reader.

After the aforementioned requirements are tested and met, the authentication is allowed to begin. All of its logic, elucidated in section 4.3, is abstracted from the user, resulting in a rather straightforward process. The `POReID middleware` prompts the user to input their PIN and, if correctly introduced, the user is authenticated successfully.

In the same way as with the administrator, a voter is identified by the public address of the account that is currently used by Metamask. However, only public addresses, whitelisted in the

voting smart contract before the configuration phase (described in section 4.2.1), are able to view and interact with this interface.

If the election has been properly configured and, consequently, started by the administrator, the voter will be greeted by the signup page, displayed in figure 4.6, in which they are asked to provide their voting keys via drag-and-drop. Once provided, the voting keys are evaluated and, if deemed correct, the user is allowed to press the `Next` button and submit them to the smart contract via a contract interaction transaction.

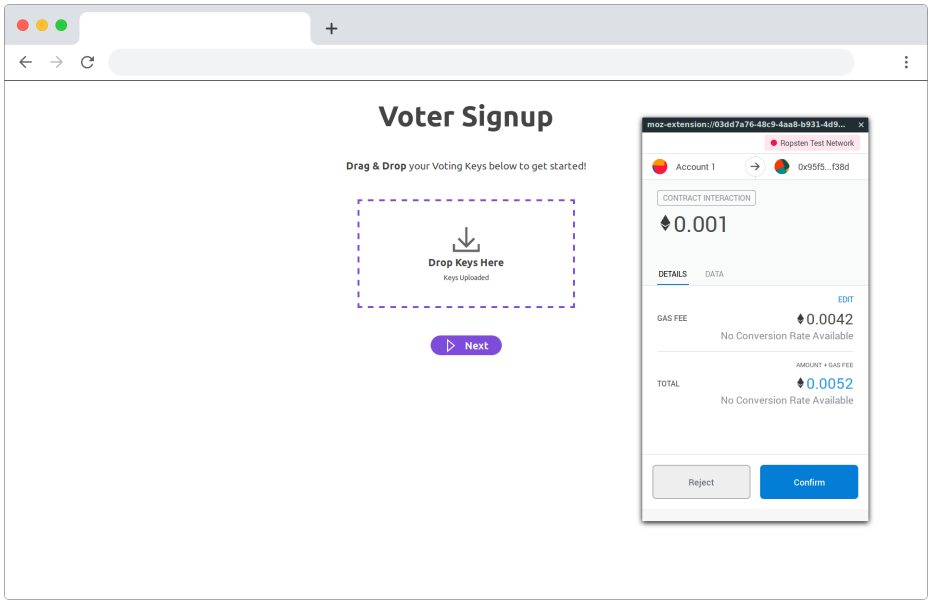

Figure 4.6: Voter registering the voting keys.

Following the aforementioned step, a view containing the total of currently registered users and the time remaining until the election can be advanced to the voting phase is displayed. This is presented on figure 4.4.

Once the election has been advanced to the voting phase, the voter is, once again, required to submit their voting keys in the same fashion as during the registration phase, displayed on figure 4.6. Afterwards, their voting keys are validated in contrast to the ones originally submitted to the smart contract and, if considered legitimate, a new view is introduced. This view, exhibited in figure 4.7, allows the voter to submit their vote by pressing the `Yes` or `No` button. Following the same approach as previous operations involving the smart contract, the voter must confirm this contract interaction in the `Metamask` window that pops-up.

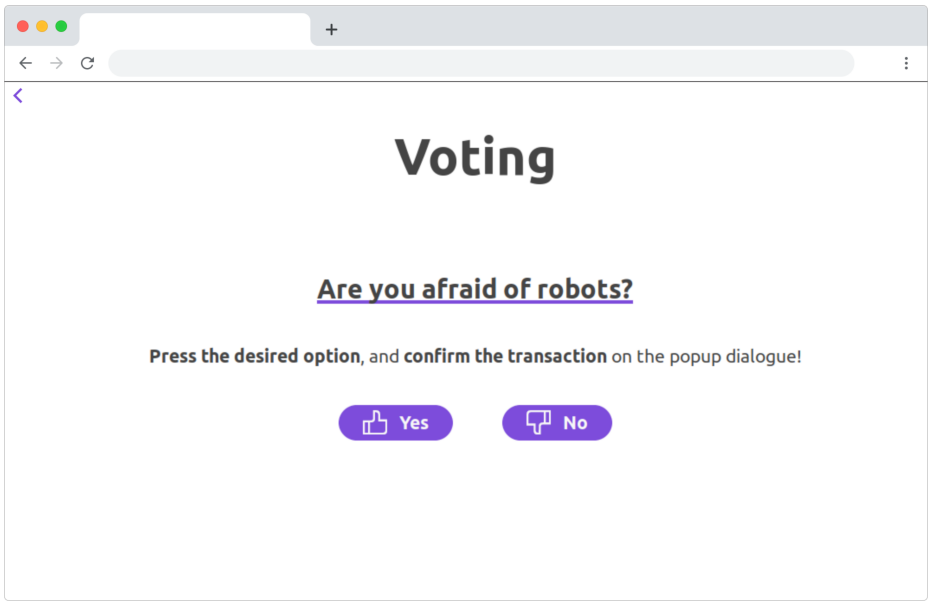

Figure 4.7: Voter casting a vote.

4.3 Implementation of the Authentication System

The authentication system is a very crucial and intricate part of the designed system. It is this scheme that allows a user to validate their identity, which will in turn allow them to interact with the voting smart contract. Solely users that have successfully authenticated themselves are allowed to have interactions with an ongoing election.

This system can be divided into three parts: (i) a local java client, (ii) a local loopback server, and (iii) a remote Node.JS sever. The client undertakes operations relating to the eID, such as obtaining data from it and signing the session token. The loopback server acts as a mediator between the Dapp and the client, in order for information to be exchanged. Lastly, the remote server validates each part of the authentication process. For instance, it verifies the correctness of the certificate chain contained in the eID, as well as the signature of the session token, generated by it.

4.3.1 Local Java Client

The client works by attending requests sent by the Dapp. These requests are obtained by constantly querying the `LoopBack API` for new entries. After a request is fulfilled, a JavaScript Object Notation (JSON) object is generated with the output result and sent as a POST request to the `LoopBack API`.

A request can be of three distinct types: `middlewareStatus`, `citizenData` and `signature`. The `middlewareStatus` request type evaluates the state of the PORelD middleware and the eID. This encompasses information related to, whether or not, the middleware is running, the eID is de-

tected and data is sucessfully read from it, and certificates are read correctly. The `citizenData` request type generates and returns a JSON object containing the ID number of the citizen, the encoded authentication certificate and a `JSONArray` containing the certificate chain. Lastly, the `signature` request type receives a token, which was generated for the current Dapp session by the Node.JS backend, and generates a signature of that token using the eID. This is accomplished in the following manner:

1. Generate a `byte[]` digest from the received token `string`;
2. Initialize a `keystore` using the instance of the citizen card;
3. Sign the digest with Secure Hash Algorithm 256 with Rivest Shamir Adleman (SHA256withRSA). The user is prompted to input the PIN of the eID, which will in turn conceive the private key used for the signature algorithm;
4. Encode the generated signature in `Base64` (the resulting signature is of the `byte[]` data type and needs to be encoded into a more human-friendly format);
5. Generate a JSON object with the encoded signature.

4.3.2 Local LoopBack API Server

A LoopBack API is a Representational State Transfer (REST) web service, which means that it uses Hypertext Transfer Protocol (HTTP) requests to `GET`, `PUT`, `POST` and `DELETE` data. As briefly discussed above, this particular API server acts as a broker for all the local communications that happen between the Dapp and the client.

This API was configured with six different model endpoints: `citizen`, `middlewareStatus`, `requestData`, `requestSignature`, `dataToSign`, and `signature`. Each of these model endpoints has an associated database table, as discussed in section 3.3.2. The server was configured to use a PostgreSQL database as the DataSource.

4.3.3 Node.JS Backend Server

The Node.JS server is responsible for verifying the authentication of the Dapp. The communications with the Dapp are fulfilled using the `socket.io` framework. This framework enables real-time bidirectional event-based communication and uses Secure Sockets Layer (SSL) to transmit information securely.

The first communication with the server happens when the Dapp begins loading. At this moment, a `sockets.io` session is established between that Dapp instance and the server, allowing back-and-forth communication using events. The server expects four different types of events: (i) `citizen-data`, (ii) `request-random-token`, (iii) `signature` and (iv) `signature-verification`.

A `citizen-data` event is responsible for associating the `sockets.io` session with the JSON containing the citizen data, sent by the Dapp, verifying that the citizen is of age to vote and that the certificate chain is valid. The certificate chain is verified in the following manner:

1. Parse the certificate chain into a JSON object and obtain the number of certificates present in the chain;
2. For every certificate in the chain, with the exception of the penultimate, the issuer of the certificate is considered to be the certificate subsequent to it in the provided chain. The following verifications ensue:
   (a) Verify that the issuer of the certificate corresponds to the certificate successive to it in the chain;
   (b) Verify that the key identifier of the certificate is correct;
   (c) Verify that the key identifier of the issuer is correct;
   (d) Verify that the signature of the issuer on the current certificate is valid.
3. Regarding the penultimate certificate, the issuer certificate is loaded from a server-stored `PEM` file, `ecraizestado.pem`, which was securely obtained from the certification authority (`https://pki.cartaodecidadao.pt`). In a similar manner, the verifications referred to in item `2` are fulfilled;
4. Verify that the first certificate is not present in any certificate revocation list, obtained securely from the aforementioned website.

If all of the verifications above complete successfully, the certificate chain is, therefore, correctly verified and an acknowledgment is sent to the Dapp using a `sockets.io` event (`valid-chain`). A better insight on how the preceding verification is accomplished, is shown in listing A.1, included in the appendix.

A `request-random-token` event makes use of the `crypto` library to generate 48 random bytes through the `randomBytes(48)` function and, afterwards, encodes the generated data into a `Base64 string`. This `string` is, in fact, the token that must be signed by the eID to prove the identity of the citizen. The token becomes associated with the current session, as well as a date by which the token must be signed by. Following this, the server emits to the Dapp, two messages using `sockets.io`: (i) `random-token`, containing the token and (ii) `token-date`, containing the expiration date of the token.

A `signature` event receives and associates the signature of the token to the session, sent by the Dapp. Following this, an acknowledgment is sent to the Dapp.

A `signature-verification` event attempts to verify the signature of the token. It accomplishes this in the following manner:

1. Load the certificate of the citizen using the `PEM string` stored in the `sockets.io` session object and extract its public key;

2. Generate a `Base64` digest of the token using Rivest Shamir Adleman (RSA)-Secure Hash Algorithm 256 (SHA256);
3. Decode the digest from `Base64` into a `Buffer` (a buffer is similar to an array of integers, but corresponds to raw memory allocation outside the heap of the Javascript v8 engine);
4. Decode the signature from `Base64` into a `Buffer`;
5. Verify the signature using the public key of the certificate, the `Buffer` of the digest and the `Buffer` of the signature using SHA256 (the SHA256 algorithm used is analogous to SHA256withRSA, used to create the signature in the local Java client).

If the signature is valid, an acknowledgment is sent to the Dapp using a `sockets.io` event (`signature-valid`). An example of the programming logic responsible for validating the signature of the session token is depicted in listing A.2, included in the appendix.

4.4 Implementation of the Ethereum Smart Contracts

The developed system makes use of two distinct Ethereum smart contracts. As previously described in section 2.2.2, these smart contracts are Turing complete programs that run on the EVM. The first of these contracts, entitled `Cryptography Contract`, a utility smart contract, is responsible for standardizing the cryptographic functions used during the voting process by the voters and the election administrator. The second of these contracts, denominated `Voting Contract`, contains the logic necessary to, among others, switch the election to different phases, verify that the election requirements are being fulfilled, and employ functions from the `Cryptography Contract` with the intent of verifying the voting keys of a given voter and calculating the final tally of the election.

4.4.1 Cryptography Smart Contract

The `Cryptography Contract` makes use of two different libraries: (i) `ECCMath` and (ii) `Secp256k1`. The first one provides basic elliptic curve math operations, such as `invmod`, with the purpose of using the euclidean algorithm to discover the modular inverse, `expmod`, responsible for calculation modular exponentiation, and `toZ1`, used internally to transform a Jacobian point $P = (Px, Py, Pz)$ to $P' = (Px', Py', 1)$. The second one contains internal versions of the `secp256k1` curve, as well as several functions necessary for performing point arithmetic. Beyond these libraries, the following functions are implemented by the `Cryptography Contract`:

- `createZKP` - creates a ZKP from the user-submitted voting keys. This function requires the x_i, v_i and g^{x_i} points of the voting keys;
- `verifyZKP` - verifies a ZKP using a user-submitted voting key. This function uses the g^{x_i} point of the voting keys;

- `create1outof2ZKPNoVote` – creates a ZKP for a negative vote, which proves that the vote is either zero or one, using the g^{x_i} and g^{y_i} points of the registered voter obtained from the `Voting Contract`, as well as w_i, r_i, d_i and x_i obtained from their voting key;
- `create1outof2ZKPYesVote` – creates a ZKP for a positive vote, which proves that the vote is either zero or one. It uses the same points as the function elucidated above;
- `verify1outof2ZKP` – verifies the correctness of the ZKP resulting from `create1outof2ZKPNoVote` or `create1outof2ZKPYesVote` against the g^{x_i} and g^{y_i} points of the voting keys.

It is worth noting that absolutely no transactions are sent to this smart contract regarding the computation of the zero knowledge proofs. These functions are read-only and, therefore, do not require any assistance from the network since there is no possibility of state change in the contract. Instead, they are executed using the local resources of the machine of the voter. An excerpt of the code responsible for interacting locally with some of the aforementioned functions, with the purpose of casting an encrypted vote to the smart contract of the election, is given in Listing A.4.

4.4.2 Voting Smart Contract

Much like the `Cryptography Contract`, the `Voting Contract` makes use of the `ECCMath` and `Secp256k1` libraries. However, differently to the preceding smart contract, this contract in particular provides methods related to the management and inner workings of the election itself. Due to the fact that the designed system is decentralized, all the data concerning the election and its processes are stored in this very smart contract and do not hinge on a private database. Instead, the memory and functions of the smart contract are publicly accessible. Due to this reason, three distinct types of users must be differentiated, with dissimilar privileges: the `administrator`, the `voter` and the `outsider`. The `administrator` is the owner of the contract by virtue of being the entity that originally deployed it to the network. There are several functions in the contract only accessible by them. This access is restricted by creating a `modifier` that verifies that the address of the user invoking the method is that of the owner of the smart contract. This modifier gets included into the header of any sensitive function and assures that no other user is able to tinker with the managerial aspect of the election. These functions are as follows:

- `setWhitelist` – whitelists the addresses of eligible users. An excerpt of the smart contract code that illustrates this function is presented in listing A.5, included in appendix A;
- `beginRegistration` – initiates the registration phase of the protocol. This function deals with the configurational aspect of the election. Such configurations are: the lifetime of each of its phases, the topic that the voting attempts to settle and the (optional) deposit fee for casting a vote. The deadline of each phase is specified in `UNIX` time and is subject to a sanity check, the objective of which is to guarantee that there are no overlaps between the different phases. The current `UNIX` time is obtained by querying the timestamp of the latest Ethereum block;

- `finishRegistrationPhase` – advances the election to the voting phase. Additionally, it reconstructs the voting keys submitted by each voter.

A `voter` is able to call any other method of the smart contract. However, there are conditions in place to prevent them from executing a certain method at an undesired time. One of these conditions is, as mentioned above, verifying that the current `UNIX` block timestamp is within one of the deadlines initially set by the `administrator`. Another way of controlling method executions is by using a modifier in its header, `inState(state)`, which only allows a method to be executed if the election is currently in the correct phase. Through these strategies, a `voter` is only ever able to interact with the smart contract in the way intended. An `outsider` is any other user that attempts to interact with the smart contract without following the protocol established by the Dapp. Because the logic and the methods of this electoral system are publicly accessible, nothing forbids someone from interacting with the smart contract without using the web application and authenticating themselves using their eID. In order to solve this, before the registration phase begins, the `administrator` must specify which addresses are able to interact with the election. These are the addresses of the citizens that successfully demonstrated to be eligible voters. As a result, non-whitelisted addresses are only ever able to ascertain the data stored in the contract and interact with the methods that have no influence over the course of the election. For instance, anyone can execute the method to compute the tally (given that enough time has elapsed for voting) and check if a deadline has been missed.

As briefly discussed above, the `administrator` initially configures the different deadlines that must be followed by everyone. These deadlines are the following:

- `endRegistrationBy` – corresponds to the deadline whereby a voter must submit their voting key by;
- `endRegistrationPhase` – concerns the time by which the admin must advance to the voting phase by;
- `endVotingPhase` – refers to the deadline whereby the tally must be calculated by;
- `endRefundPhase` – corresponds to the time by which a voter must ask for a refund by.

In the eventuality that the admin does not comply with the established deadlines, the voters have the possibility of requesting a refund for their deposit fee. This refund does not require the permission of the admin. If the smart contract ascertains that a deadline was, indeed, not fulfilled, the voter is refunded without delay, provided that they are within the refund period.

4.5 Conclusions

This chapter described the implementation details regarding each individual module of the e-voting system, as well as the different interactions between them. In general, the central-most module of the system may be considered to be the Dapp, which is responsible for the interactions

of the voter or the election administrator with the system. The Dapp communicates with the eID of the voter through a local Java client and evaluates the authentication and the validity of the certificates present in the eID by consulting with a remote server. Lastly, the Dapp simplifies the electoral process by interacting with the publicly verifiable Ethereum smart contracts that hold the logic of the election.

In essence, the developed system abstracts the complexity of the underlying system from its users, while providing individual and universal verifiability, eligibility, fairness and accuracy (these concepts are described in section 3.2.1).

The implementation of the authentication system presented many challenges, namely in assessing a way to interact with the eID of the citizen, since the official support to developers makes use of a Java browser plug-in. These plug-ins have been considered to be extremely insecure [19], being even officially deprecated by Oracle in Java Development Kit (JDK) 9 [20]. Due to this reason, the authentication system had to be built from the ground up, by implementing an authentication API, a local client that interprets and executes the various requests sent by the Dapp, and a remote server that verifies the legitimacy of the authentication and certificates. This client, in particular, made use of the Software Development Kit (SDK) of the Portuguese citizen card in order to interact with the many functions required for the authentication mechanism.

The development of the Dapp also proved to be quite a challenging task. The reason for this was the novelty of the technologies used — mainly the Ethereum smart contracts and the `Web3.JS` library — as well as the lack of correct and updated documentation supporting them.

Chapter 5

Fulfillment of the Security Requirements, Testing and Financial Breakdown

5.1 Introduction

This chapter elaborates on the fulfillment of the security requirements, previously identified in section 3.2, in section 5.2. Moreover, several user tests and their respective results are specified and discussed in section 5.3. Finally, the financial aspect of running an election using the proposed embodiement of the voting scheme is analyzed in section 5.4.

5.2 Fulfillment of the Security Requirements

As previously discussed in section 3.2.1, an electoral system comprises a set of rules and protocols that govern the different facets of the voting process. These rules and protocols may be encapsulated into a handful of security requirements, namely: **accuracy**, **privacy**, **individual and universal verifiability**, **eligibility** and **fairness** [5]. Table 2.1 displays the fulfillment, or lack thereof, of these security requirements by several works. Particularly, our project conforms with every single requirement by employing the technologies described on table 2.2 in the following manner:

- **Privacy** — Protocols that rely on a central authority to verify the eligibility of voters and, proven eligible, attribute them voting keys, embody a point of failure to the system itself, as this central authority may be able to link a voter to its vote. This issue was rectified by employing the Portuguese eID to validate the eligibility and authenticity of a voting citizen. This government-issued citizen card allows for the authentication of a voter through the usage of their digital signature. Furthermore, it is possible to verify that the digital certificate chain of a given card is valid, as well as certify that the higher-level certificate (relating to the card itself) has not been revoked;
- **Accuracy** — By virtue of the *immutability* of the Ethereum Blockchain [6] and the logic present on the voting smart contract used by the Dapp, a voter is not able to alter their vote;
- **Fairness** — Due to the logic present in the smart contract, the elector is able to compute the tally by oneself once the voting phase of the election has elapsed, assuring that the results are not revealed prior to the end date;

- **Eligibility** — Through the usage of certificate revocation lists, each citizen may only have one valid eID at any given time. In view of this fact, a citizen is not able to recurrently cast a vote:
- **Individual and Universal Verifiability** — By adopting the voting protocol developed by Hao *et al.* [2], the voting citizen is able to verify that their vote has been recorded as cast, as well as corroborate that the remaining voters act according to the protocol.

5.3 User Tests Apparatus and Results

Specific tests were created regarding the usage of the e-Voting Dapp. These tests pertained to both the administrator and the voter workflow, depicted previously in figure 3.1 and in figure 3.2, respectively.

The user tests of the Dapp were performed in-person, over a group of 10 people with different ages, gender and educational backgrounds. The test group was then split into two subgroups, each comprised of five test subjects. Each one of the subjects enacted the role of administrator and the role of voter, which resulted in a total of five elections, for each subgroup. Regarding each of these elections, the subject portraying as the administrator was required to undertake the activities required to deploy, configure and manage an election, while the subjects portraying as voters were required to perform the activities related to registering to an election and casting a vote. Before the testing began, the testees were lightly briefed regarding what type of application they would encounter, and the different roles that they would portray. During each iteration, the user acting as administrator was given further instructions regarding the choosing of the different deadlines. After the testing was concluded, the test subjects were interviewed, individually, regarding the usability of the system. This interview consisted of five different user-experience questions, to be rated on a scale of 1-5:

1. How simple was portraying the role of administrator?
2. How simple was portraying the role of voter?
3. How secure did you feel by voting in this manner, comparing to traditional voting?
4. How interested would you be in using this application in the future?
5. Visually, how appealing do you consider the application?

Table 5.1 presents the results of the questions asked to each test subject. Furthermore, some testees complemented certain answers with an observation. One common remark mentioned in the scope of question three, was that, despite understanding the security requirements and its fulfillment, the users did not have the scientific and technological know-how to verify the legitimacy regarding the claims of resistance to electoral fraud. This prevailing opinion could present itself to be an obstacle in a future real-world implementation of this scheme. Nevertheless, the ability to vote directly over their personal computer using a Portuguese citizen card was referred to be "*tremendously advantageous*". Regarding question five, subjects reported that

Question \ Rating	1	2	3	4	5
1			2	4	4
2				3	7
3		1	4	4	1
4				2	8
5				3	7

Table 5.1: The compounded rating of all subjects regarding each UX question.

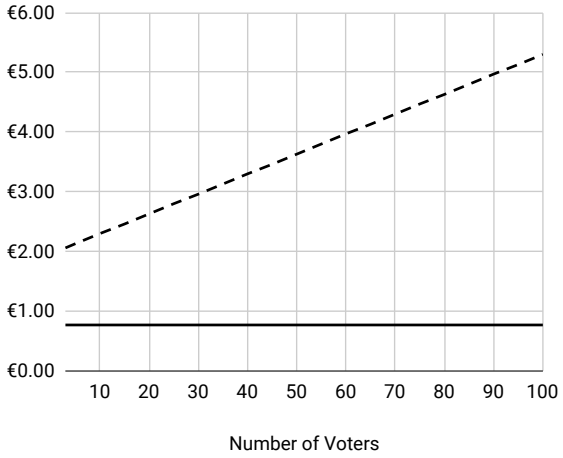

Figure 5.1: The financial breakdown, in Euro, of an election, depending on the number of voters for both the administrator (dashed) and voters.

Entity: Transaction	Cost in Gas	Cost in €
A: Deploy Voting Contract	4,719,005	0.99
A: Deploy Crypto Contract	4,293,778	0.91
A: Set Eligible Addresses	2,201,111	0.46
A: Begin Signup	189,235	0.04
V: Register	3,369,366	7.10
A: Begin Election	3,500,037	0.74
V: Vote	111,701,825	23.55
A: Tally	741,868	0.16
A: Total	15,645,034	3.30
V: Total	145,395,485	30.65
Election Total	161,040,519	33.95

Table 5.2: The financial breakdown of the costs associated with running an election featuring 40 voters. The costs, expressed in EUR (€), entail a conversion rate of 1 ETH = €210.18 (accurate as of 6 June 2019) and a cost per unit of gas of 1 Gwei.

the minimalist layout of the application and its simplified use cases resulted in a very smooth experience.

5.4 Financial Breakdown

Figure 5.1 displays the financial breakdown, in Euro, of the election administrator and each voter, depending on the total number of voters. As can be observed, it scales linearly for the administrator, with an added cost of €0.17 for each additional voter, and a static cost of €0.77 for each individual voter. Nevertheless, it has been noted that, due to computational restrictions on the Ethereum Blockchain (relating to the cost in gas for different operations), there is an upper limit of 90 voters in the current iteration of this particular voting protocol. Table 5.2 demonstrates the gas costs for the different types of transactions required for running an election. The cost of elections is currently volatile, but will potentially decrease with the transition to PoS.

Figures for the Portuguese Legislative Election of 2019 (according to Isabel Oneto, Secretary of State Assistant and of Internal Administration [21]), show that the expected operational cost

for the elections for individuals in Portugal and abroad was of 9 Million and 7 Million euro, respectively. Therefore, the cost of each vote (considering an equivalent number of electors as in the Portuguese Legislative Election of 2015 [22]) is of €1.67 and €28.82. Though these are estimates, they provide a rough idea that the proposed system may also comprise a financial incentive to a real-world implementation.

The amount of voters supported by the voting protocol can be largely increased by batching transactions during the voting phase, or be potentially limitless if executed on a private Blockchain (which would be publicly scrutinized). In terms of infrastructure, the Blockchain could be installed and managed by public institutions such as universities.

5.5 Conclusions

This chapter discussed how the security requirements determined initially were realized. The conclusion was drawn that all the security requirements were satisfied. Furthermore, all the functionalities of the Dapp were tested by two different test groups which were interviewed regarding their experience at the end. Through this interview, it was possible to extrapolate results regarding the user experience of the voting system, as well as draw conclusions regarding future improvements. The user experience was predominantly positive. Concerning the financial breakdown, the developed system proved to be financially superior to a traditional voting system, especially when considering voting from abroad. Furthermore, the financial cost may be decreased significantly with future upgrades to the Ethereum ecosystem.

Chapter 6

Conclusions and Future Work

This final chapter presents the main conclusions of the work described in this dissertation in section 6.1, and points out potential supplementary functionalities that may be used to improve this work, in the future, in section 6.2.

6.1 Main Conclusions

This dissertation has presented a Blockchain e-voting system that significantly improves current schemes by making use of government-issued eIDs to authenticate citizens and verify their eligibility to vote. The voting protocol used allows for E2E auditability and maximum voter privacy due to the usage of zero-knowledge proofs and its execution being safeguarded by the same consensus mechanism that protects the Ethereum Blockchain. Furthermore, and by virtue of the *immutability* of the Ethereum Blockchain, anyone can verify that the rules of the protocol are enforced justly.

This work was initially divided in the following objectives:

1. Study the security challenges associated with electronic voting systems; identify requirements (namely security requirements);
2. Engineer an electronic voting system; implement a set of Ethereum smart contracts and decentralized apps underlying the particular case studied in the scope of this project;
3. Implement a web application that makes use of the Portuguese Citizen Card eID, for authentication purposes, and that will allow the user to cast a vote, without knowledge of the underlying system.

All of these objectives were successfully accomplished. Objective number **one** was realized by studying, specifying and explaining the different requirements pertaining to e-voting in a section named **E-Voting System Requirements** (section 3.2). Objective number **two** was achieved by, initially, identifying the software system requirements (described in section 3.3), followed by the crafting of the proposal for the system (illustrated in section 3.4). Objective number **three** was accomplished by accordingly implementing the proposal for the system in a chapter named **System Implementation** (chapter 4).

The various user tests carried out regarding the usability of the Dapp indicate the user experience to be predominantly positive. Also, the analysis of the costs associated with running an

election suggest that this voting system could comprehend a financial incentive to a traditional voting scheme.

Succinctly, the proposed system provides several improvements to traditional voting and e-voting systems. In point of fact, all the core properties of an e-voting system (described in section 3.2.1) and the additional underlying properties (essentially **robustness**, **mobility**, **integrity** and **convenience**) were assured. The security, privacy, convenience and financial incentives of this protocol could prove to be beneficial to an upsurge in voter turnout and the credibility of the electoral democratic system itself.

6.2 Future Work

Despite the main objectives of this work being fulfilled, there are several functionalities that may be added to the system in pursuance of additional security, convenience and performance. These functionalities are as follows:

- Validate the certificate-chain present in the eID and verify the eligibility of the voting citizen using a decentralized network of oracles (in the prototype of the system, the validation of the chain is still performed centrally). This network of oracles would follow a protocol that incentivizes each node to tell the truth, with active penalties for acting dishonestly;
- Extend the protocol to support multiple candidates (currently, it only accepts two voting options), following the approach disclosed in [2]. Doing so would, however, increase the computational load per participant by k times (k refers to the number of candidates). The decision of choosing the number of candidates and the consequential decrease in performance would be up to the administrator of the election.
- Extend support in the Dapp to Metamask alternatives — such as Fortmatic — which does not require the user to download a browser extension. As the technical challenges of new paradigms prove to cause friction with adoption, the onboarding of users and the required knowledge of the underlying system must be streamlined;
- Batch smart contract operations, in order to reduce financial and computational costs. As explained in section 5.4, an Ethereum smart contract only allows a certain amount of computations to be executed for any given smart contract call. Therefore, there is currently an upper limit in the vicinity of 90 voters. This limit is due to the voting keys being computed in a single transaction. By batching the work of this operation in various transactions, this restriction may be removed;
- Store voting keys in the internal memory of the eID, in furtherance of convenience to the voting citizen. In order to do this safely, however, the voting keys must be encrypted and decrypted using an asymmetric key encryption scheme. Unfortunately, using the private key of the Portuguese citizen card for anything other than signing is restricted. This presents a limitation to what can be accomplished using the eID.

Bibliography

[1] J. D. Monteiro, B. Sequeiros, M. Freire, and Pedro R. M. Inácio, "Blockchain-based decentralized application for electronic voting using an electronic id," in *Atas do INForum 2019*. NOVA.FCT Editorial, 2019. [Online]. Available: http://inforum.org.pt/INForum2019/docs/atas-do-inforum2019 viii, 5

[2] F. Hao, P. Ryan, and P. Zielinski, "Anonymous voting by two-round public discussion," *IET Information Security*, vol. 4, no. 2, p. 62, 2010. [Online]. Available: https://doi.org/10.1049/iet-ifs.2008.0127 ix, xi, 10, 11, 40, 44

[3] "Blockchain voting: The end to end process." [Online]. Available: https://followmyvote.com/blockchain-voting-the-end-to-end-process/ ix, 11

[4] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography and Data Security*. Springer International Publishing, 2017, pp. 357–375. [Online]. Available: https://doi.org/10.1007/978-3-319-70972-7_20 ix, 10, 11, 17

[5] R. Anane, R. Freeland, and G. Theodoropoulos, "e-voting requirements and implementation," in *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*. IEEE, jul 2007. [Online]. Available: https://doi.org/10.1109/cec-eee.2007.42 ix, 12, 15, 39

[6] V. Buterin, "ethereum/wiki," Dec 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper 7, 17, 39

[7] A.-G. Tsahkna, "E-voting: Lessons from estonia," *European View*, vol. 12, no. 1, pp. 59–66, Jun. 2013. [Online]. Available: https://doi.org/10.1007/s12290-013-0261-7 8

[8] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security analysis of the estonian internet voting system," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*. ACM Press, 2014. [Online]. Available: https://doi.org/10.1145/2660267.2660315 8, 12

[9] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*. ACM Press, 1985. [Online]. Available: https://doi.org/10.1145/22145.22178 9

[10] A. Kiayias and M. Yung, "Self-tallying elections and perfect ballot secrecy," in *Public Key Cryptography*. Springer Berlin Heidelberg, 2002, pp. 141–158. [Online]. Available: https://doi.org/10.1007/3-540-45664-3_10 10, 11

[11] J. Groth, "Efficient maximal privacy in boardroom voting and anonymous broadcast," in *Financial Cryptography*. Springer Berlin Heidelberg, 2004, pp. 90–104. [Online]. Available: https://doi.org/10.1007/978-3-540-27809-2_10 10

[12] E. Yavuz, A. K. Koc, U. C. Cabuk, and G. Dalkilic, “Towards secure e-voting using ethereum blockchain,” in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, Mar. 2018. [Online]. Available: https://doi.org/10.1109/isdfs.2018.8355340 10

[13] TIVI, “Online voting - successfully solving the challenges,” Whitepaper, 2017. [Online]. Available: https://www.smartmatic.com/fileadmin/user_upload/Whitepaper_Online_Voting_Challenge_Considerations_TIVI.pdf 11

[14] L. Sheeter, “Estonia forges ahead with e-vote,” Oct 2005. [Online]. Available: http://news.bbc.co.uk/2/hi/europe/4343374.stm 11

[15] A. Charlton and K. de Pury, “Election fraud in russia caught on video: ballot-stuffing, erasable ink and more,” Dec 2011. [Online]. Available: https://observers.france24.com/en/20111206-russia-election-fraud-caught-video-ballot-stuffing-erasable-ink-putin-protests 17

[16] T. P. C. Trusts, “Evidence That America’s Voter Registration System Needs an Upgrade,” 2012. [Online]. Available: www.pewcenteronthestates.org/ELECTIONS 17

[17] D. Butler, “Electoral reform,” *Parliamentary Affairs*, vol. 57, no. 4, p. 735, Oct. 2004. [Online]. Available: https://doi.org/10.1093/pa/gsh057 17

[18] A. Renwick, “Electoral reform in europe since 1945,” *West European Politics*, vol. 34, no. 3, pp. 456–477, May 2011. [Online]. Available: https://doi.org/10.1080/01402382.2011.555975 17

[19] J. S. Fritzinger and M. Mueller, “Java security,” *White Paper, Sun Microsystems, Inc*, 1996. 37

[20] D. Topic, “Moving to a plugin-free web,” Jan 2016. [Online]. Available: https://blogs.oracle.com/java-platform-group/moving-to-a-plugin-free-web 37

[21] Lusa, “Voto dos portugueses no estrangeiro vai custar sete milhões de euros,” Sep 2018. [Online]. Available: https://www.publico.pt/2018/09/29/politica/noticia/voto-dos-portugueses-no-estrangeiro-vai-custar-sete-milhoes-de-euros-1845697 41

[22] SGMAI-AE and BDRE, “Resultado do Recenseamento Eleitoral - Eleições Legislativas,” Sep 2015. [Online]. Available: https://www.sg.mai.gov.pt/AdministracaoEleitoral/EleicoesReferendos/AssembleiaRepublica/Documents/AR2015_Contagem_Eleitores_20150919.xls 42

Appendix A

Code Excerpts

This chapter presents some of the lengthier code excerpts mentioned in the body of the dissertation. They were included in order to provide further details on the implementation presented in chapter 4, mainly in regards to the validation of the authentication of the voter, the communication with the LoopBack API by means of POST requests, and the process of submitting the encrypted vote to the election smart contract.

```
verifyCertChain = (chain) => {
    const parsedChain = JSON.parse(chain)
    const numberOfCerts = parsedChain.length

    for (let i = 0; i < numberOfCerts - 1; i++) {
        //Verify ecraizestado cert manually on last iteration
        let issuer = null
        if (i === numberOfCerts - 2)
            issuer = Certificate.fromPEM(fs.readFileSync("./cc_certificates/
                ecraizestado.pem"))
        else
            issuer = Certificate.fromPEM(parsedChain[i + 1].cert)

        let cert = Certificate.fromPEM(parsedChain[i].cert)

        let isIssuer = cert.isIssuer(issuer) // true
        let issuerKeyIdentifierVerified = issuer.verifySubjectKeyIdentifier()
            // true
        let certKeyIdentifierVerified = cert.verifySubjectKeyIdentifier() //
            true
        let checkedSignature = issuer.checkSignature(cert) //null

        if (!isIssuer) {
            console.log("ABORT!Invalid Issuer.")
            return false
        }

        if (!issuerKeyIdentifierVerified) {
            console.log("ABORT!Invalid Issuer Key Identifier.")
            return false
        }

        if (!certKeyIdentifierVerified) {
```

```
            console.log("ABORT!Invalid Cert Key Identifier.")
            return false
        }

        if (checkedSignature !== null) {
            console.log("ABORT!Invalid Signature.")
            return false
        }
        console.log("Cert #" + i + " OK")
    }
    return true
}
```

Listing A.1: Excerpt of code that verifies the validity of the certificate chain pertaining to the electronic ID of a voter.

```
client.on("signature-verification", () => {
    console.log("\nClient [" + client.id + "] Requested Signature Verification")
    const signature = currentConnections[client.id]._signature
    const cert = Certificate.fromPEM(currentConnections[client.id].
        _citizen_data.cert)
    const pubKey = cert.publicKey
    const token = currentConnections[client.id]._token

    // Digest in byte[] JS equivalent Create a RSA-SHA256 with UTF-8 input
        encoding
    // and digest it to base64
    let digest = crypto
        .createHash("RSA-SHA256")
        .update(token, "utf8")
        .digest("base64")

    // Decode base64 digest and create a buffer (byte[] => Uint8Array)
    byteArray_digest = Buffer.from(digest, "base64")

    // Signature in byte[] JS equivalent; Decode base64 signature and create a
        buffer
    // (byte[] => Uint8Array)
    let byteArray_sig = Buffer.from(signature, "base64")

    // Verify signature using the public key of the certificate, the byte[]
        digest and the signature
    // Sha256 is analogous to the Sha256WithRsa signature created in the Java
        client
    let result = pubKey.verify(byteArray_digest, byteArray_sig, "sha256")
    console.log("Signature Valid: " + result)
    client.emit("signature-valid", result)
```

```
27 });
```

Listing A.2: Excerpt of code that validates the signature of the token generated by the electronic ID of a voter.

```
1  genericPostRequest = async(endpoint, data) => {
2      const url = API + endpoint
3      const date = new Date().getTime()
4      data["date"] = date.toString()
5      console.log("Data: " + JSON.stringify(data))
6      const response = await fetch(
7          url,
8          {
9              method: "POST",
10             headers: {
11                 'Accept': 'application/json',
12                 'Content-Type': 'application/json',
13         },
14         body: JSON.stringify(data)
15     })
16
17     const json = await response.json()
18     return json
19 }
```

Listing A.3: Excerpt of code that depicts how POST requests are conceived by the Dapp and sent to the according endpoint of the local LoopBack API server.

```
1  handleCastVote = async (choice) => {
2
3      // Check in the election contract if the user is registered to vote
4      let registered = await this.props.anonContract.methods.registered(
5          this.props.metamaskAddress).call()
6
7      if(!registered)
8          return
9
10     let voter = await this.props.anonContract.methods.getVoter()
11                     .call({from: this.props.metamaskAddress})
12     let result = undefined
13
14     let xG = [voter[0][0], voter[0][1]]
15     let yG = [voter[1][0], voter[1][1]]
16
17     if (choice == "YES")
18         // Create 1-of-2 ZKP locally
19         result = await this.props.cryptoContract.methods.
               create1outof2ZKPYesVote(
20             xG,
```

```
            yG,
            this.state.parsedFile.w,
            this.state.parsedFile.r,
            this.state.parsedFile.d,
            this.state.parsedFile.x).call({from: this.props.metamaskAddress})
    else
        // Create 1-of-2 ZKP locally
        result = await this.props.cryptoContract.methods.create1outof2ZKPNoVote
            (
            xG,
            yG,
            this.state.parsedFile.w,
            this.state.parsedFile.r,
            this.state.parsedFile.d,
            this.state.parsedFile.x).call({from: this.props.metamaskAddress})

    let y = [result[0][0], result[0][1]]
    let a1 = [result[0][2], result[0][3]]
    let b1 = [result[0][4], result[0][5]]
    let a2 = [result[0][6], result[0][7]]
    let b2 = [result[0][8], result[0][9]]

    let params = [result[1][0], result[1][1], result[1][2], result[1][3]]

    // Verify locally that 1-of-2 ZKP is valid
    result = await this.props.cryptoContract.methods.verify1outof2ZKP(
                    params, xG, yG, y, a1, b1, a2, b2).call({from: this.props.
                        metamaskAddress})

    if(result === false) {
        this.setState({
            status: "error",
            statusMsg: "Invalid ZKP"})
        return
    }

    // Submit encrypted vote to the smart contract
    result = await this.props.anonContract.methods.submitVote(
                    params, y, a1, b1, a2, b2)
                    .send({from: this.props.metamaskAddress, gas: 4200000})
```

Listing A.4: Excerpt of code that presents how the Dapp creates, validates and submits a 1-of-2 ZKP containing the vote to the smart contract of the election.

```
1     // This modifier enforces that only the owner of the contract may call a
          function decorated with it.
2     modifier onlyOwner {
3         if(owner != msg.sender) throw;
4         _;
5     }
6
7     // The owner of the contract (the administrator of the election) configures
           the addresses that are eligible to vote. The modifier "onlyOwner" is
          responsible for enforcing that no other address may interact with this
          function.
8     function setWhitelist(address[] a) onlyOwner {
9
10    // There is an upper limit of 90 voters in the current iteration of this
          particular voting protocol
11    if((totalEligible + a.length) > 90) {
12      throw;
13    }
14
15    // Store, in the memory of the smart contract, the addresses sent by the
          administrator of the election.
16    for(uint i = 0; i < a.length; i++) {
17
18        if(!eligible[a[i]]) {
19            eligible[a[i]] = true;
20            addresses.push(a[i]);
21            totalEligible += 1;
22      }
23    }
24  }
```

Listing A.5: Excerpt of code that presents the smart contract logic related to the whitelisting of addresses.