UNIVERSIDADE D
COIMBRA

Leandro Emanuel Almeida Rodrigues

IMPLEMENTAÇÃO DE UMA RESTFUL API PARA VOTAÇÃO ELETRÓNICA

UMA PLATAFORMA PARA PARTICIPAÇÃO PÚBLICA

Dissertação no âmbito do Mestrado em Segurança Informática orientada pelo Professor Doutor Fernando Boavida e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2020

Esta página foi deixada em branco propositadamente.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao meu orientador, o Professor Doutor Fernando Boavida, por ter aceite o convite para orientar esta dissertação, pela sua disponibilidade e pelos sábios e oportunos conselhos ao longo da realização da presente dissertação. Agradecer à entidade acolhedora, a Libertrium Lda, pelo incentivo para participar na criação de um sistema de votação eletrónica remota seguro para integrar com as suas plataformas.

Agradecer ao meu orientador da entidade acolhedora, o Engenheiro Pedro Agante, por, nos momentos em que foi necessário, ter manifestado o seu apoio e contribuído com o seu conhecimento para levar este trabalho a bom porto. Ao Professor Doutor Artur Sousa, o meu agradecimento pela oportunidade, pelo convite para participar neste projeto e por todo o apoio prestado.

Por fim, mas não menos importante, agradecer à minha companheira de longa data pelo seu apoio incondicional, pela força que me transmitiu, pela motivação e pelas inúmeras validações e correções do Português. A ti o meu maior e sentido agradecimento. À minha família, em especial aos meus pais e irmã, que durante todo o meu percurso académico manifestaram todo o seu apoio e me deram, nos momentos mais complicados, o alento para nunca desistir.

Aos meus colegas e amigos um sentido agradecimento pelas partilhas, pela amizade e pela força que me deram para concluir esta dissertação. Talvez mesmo sem se aperceberem o Vosso apoio e compreensão foi a terra firme onde me sustentei nos momentos mais complicados.

Esta página foi deixada em branco propositadamente.

Abstract

Throughout the world, participatory budgets, particularly in democratic countries, have taken on an important role. They are participation instruments that allow citizens to propose, decide, or influence how budget funds (governmental or municipal) are used. Through voting, citizens have been called to participate in this process. Nowadays, where constant mobility is required, the exercise of the vote is difficult to be carried out by traditional systems. With the widespread use of technologies and access to the internet, it is believed that technology can overcome this difficulty.

Over the years, several proposals have been made for increasing the security of electronic voting systems and ensuring compliance with the various associated requirements. Helios and EVIV are examples of these proposals that, through mechanisms, techniques, and cryptographic algorithms, associated with a good deal of engineering, take security to a new level allowing for the strong mobility observed today and taking advantage of the characteristics of the ubiquity of the internet. It should be noted, however, that building a secure system, which is considered tamper-resistant, is, in our opinion, impossible.

The main objective of this thesis is to analyze some contributions to secure electronic voting, to identify one that is suitable for voting in participatory budgets, and to implement a Restful API, based on that proposal, which can integrate with the current participatory budgeting platforms from which the host entity is responsible. In this sense, there will be two proposals, aimed at remote electronic voting, which are considered to be the most promising for what it is intended to build.

Throughout this thesis, we will address the various types of voting systems, their requirements, and how they try to guarantee compliance, the most common types of voting, and cryptographic primitives that are used, as a rule, for the construction of this type of systems. We will also do a brief analysis of some proposed electronic voting systems. Participatory budgets, their functioning, their phases, and their rules will be also analyzed.

Finally, at this thesis, we will also present the functional and non-functional requirements for the solution to develop. Describing the implemented solution, some of the functional and non-functional tests will be also addressed. To conclude the thesis, we summarise the contributions and identify guidelines for future work.

Keywords

Voting Systems, Eletronic Voting, Participatory Budgets, RESTfull API, Homomorphic Encryption, Zero-Knowledge Proofs

Esta página foi deixada em branco propositadamente.

Resumo

Por todo o mundo os orçamentos participativos, em especial nos países democráticos, têm assumido um papel importante. São instrumentos de participação que permitem aos cidadãos propor, decidir ou influenciar a forma como são utilizados os fundos orçamentais (governamentais ou municipais). Através do voto, os cidadãos têm vindo a ser chamados a participar neste processo. Nos dias atuais, em que nos é exigida uma constante mobilidade, o exercício desse direito torna-se difícil de ser realizado pelos sistemas tradicionais. Com a massificação das tecnologias e também do acesso à internet, acredita-se que a tecnologia poderá colmatar esta dificuldade.

Ao longo dos anos, várias propostas foram feitas com vista a aumentar a segurança dos sistemas de votação eletrónicos remotos e a garantir o cumprimento dos diversos requisitos associados. O Helios e o EVIV são exemplos dessas propostas que, através de mecanismos, técnicas e algoritmos criptográficos, associados a uma boa dose de engenharia, elevam a segurança a um novo patamar, permitindo a forte mobilidade verificada nos dias de hoje e tirar partido das características de ubiquidade da Internet. Note-se, no entanto, que construir um sistema totalmente seguro, que seja considerado totalmente inviolável, é, na nossa opinião, impossível.

O principal objetivo desta dissertação é analisar algumas contribuições feitas para votação eletrónica segura, identificar uma que seja adequada à votação em orçamentos participativos e implementar uma *RESTful API*, a partir dessa mesma proposta, que possa integrar com as atuais plataformas de orçamento participativo, das quais a entidade acolhedora é responsável. Neste sentido serão apresentadas duas propostas, direcionadas à votação eletrónica remota, que se consideram ser as mais promissoras para o que se pretende construir.

Ao longo desta dissertação abordaremos os vários tipos de sistemas de votação, os seus requisitos e a forma como se tenta garantir o seu cumprimento, os tipos de votação mais comuns e as primitivas criptográficas às quais se recorre, por norma, para a construção deste tipo de sistemas. Serão ainda analisados alguns sistemas de votação eletrónica propostos e, neste sentido, far-se-á uma breve análise dos mesmos. Os orçamentos participativos, o seu funcionamento, as suas fases e regras serão também abordados.

Para finalizar, nesta dissertação serão também apresentados os requisitos funcionais e não funcionais da solução a desenvolver. Descrever-se-á a implementação da solução, mais concretamente da *RESTful API*, os testes funcionais e não funcionais realizados e, no final, apresentam-se as conclusões do trabalho desenvolvido e o trabalho futuro.

Palavras-chave: Sistemas de Votação, Votação Eletrónica, Orçamentos Participativos, RESTfull API, Criptografia Homomórfica, Zero-Knowledge Proofs

Esta página foi deixada em branco propositadamente.

Conteúdo

1 Introdução **1**
1.1 Motivação . . . 2
1.2 Objetivos da proposta . . . 2
1.3 Contribuições . . . 3
1.4 Organização do documento . . . 3

2 Sistemas de votação **6**
2.1 Requisitos Gerais Votação Tradicional em Urna . . . 6
2.2 Requisitos Gerais Votação Eletrónica . . . 7
2.3 Sistemas de Votação Tradicionais . . . 11
2.4 Sistemas Eletrónicos Presenciais . . . 13
2.5 Sistemas Eletrónicos Remotos . . . 13
2.6 Tipos de votação . . . 15
2.7 Conclusão . . . 18

3 Primitivas criptográficas **19**
3.1 Mix networks . . . 19
3.2 Criptografia Homomórfica . . . 20
3.2.1 Tipos de encriptação homomórfica . . . 22
3.2.2 ElGamal . . . 22
3.3 Blind Signatures . . . 24
3.4 Zero-Knowledge Proofs . . . 26
3.4.1 Métodos Interativos . . . 26
3.4.2 Métodos não interativos . . . 27
3.4.3 Casos de uso . . . 28
3.5 Conclusão . . . 28

4 Análise de Sistemas de Votação Eletrónica **30**
4.1 Scantegrity II . . . 30
4.2 Helios Voting . . . 35
4.3 EVIV . . . 39
4.4 Conclusão . . . 42

5 Orçamentos participativos **44**
5.1 Fases de um orçamento participativo . . . 44
5.2 Sistemas de votação . . . 45
5.3 Caderno eleitoral . . . 45
5.4 Regras de votação . . . 46
5.5 Funcionamento do processo de votação . . . 47
5.6 Conclusão . . . 48

6 Especificação da solução **49**

6.1 Análise de requisitos . . . 49
6.1.1 Funcionais . . . 49
6.1.2 Não funcionais . . . 52
6.2 Enquadramento . . . 52
6.2.1 Principais riscos aplicacionais . . . 52
6.2.2 Arquiteturas *single tenant* e *multi tenant* . . . 54
6.2.3 Disponibilidade de serviço . . . 55
6.2.4 Autenticação . . . 57
6.3 Arquitetura base . . . 59
6.3.1 Atores . . . 61
6.3.2 Linguagem de programação e *framework* . . . 62
6.3.3 Biblioteca Helios . . . 62
6.4 Conclusão . . . 63

7 Implementação da *RESTful API* 64
7.1 Autenticação dos gestores eleitorais . . . 64
7.2 Alteração dos algoritmos de hashing do Helios . . . 64
7.3 R1 - Criação de eleições . . . 65
7.4 R2 - Edição de eleição . . . 66
7.5 R6 - Registar eleitor . . . 68
7.6 R8 - Registar *trustee* . . . 69
7.7 R9 - Eliminar *trustee* . . . 71
7.8 R11 - Congelar eleição . . . 71
7.9 R14 - Adição de voto encriptado . . . 73
7.10 R15 - Adição de voto . . . 75
7.11 R16 - Obter detalhe de um voto . . . 76
7.12 R21 - Contabilizar os votos . . . 77
7.13 R22 - Submeter os *decryption factors* e as *decryption proofs* dos *trustees* . . . 78
7.14 R23 - Combinar os *decryption factors* dos *trustees* . . . 80
7.15 R24 - Submeter voto auditado . . . 81
7.16 Conclusão . . . 83

8 Validação da proposta 84
8.1 Testes funcionais . . . 84
8.1.1 Criação de eleição sem indicação de parâmetros . . . 85
8.1.2 Criação de eleição . . . 86
8.1.3 Adicionar um trustee . . . 87
8.1.4 Adição de um eleitor à eleição . . . 88
8.1.5 Adição de um eleitor à eleição com o mesmo identificador . . . 90
8.1.6 Congelamento de uma eleição . . . 90
8.1.7 Adição de um voto encriptado . . . 92
8.1.8 Adição de um voto encriptado inválido . . . 94
8.1.9 Contabilização dos votos . . . 95
8.1.10 Combinação dos resultados parciais e desencriptação dos resultados . . . 96
8.2 Testes não funcionais . . . 97
8.2.1 Criação de eleições . . . 98
8.2.2 Adição de eleitores a eleições . . . 99
8.2.3 Congelamento de eleições . . . 100
8.2.4 Adição de votos encriptados . . . 102
8.2.5 Contabilização de votos encriptados . . . 103
8.2.6 Desencriptar resultado da eleição . . . 104
8.3 Conclusão . . . 105

9 Conclusão **106**
9.1 Trabalho futuro 107

Esta página foi deixada em branco propositadamente.

Acrónimos

API Application programming interface. 64, 66–68, 70, 115, 117, 119, 121

BPS Ballot Preparation System. 37

DDOS Distributed Denial-of-service. 14, 56

DOS Denial-of-service. 14

DRF Django Rest Framework. 84, 85

E2E End to End. 42

E2E-V End to End Verifiable. 10, 13, 14, 18, 30, 35, 39, 42, 43

FCTUC Faculdade de Ciências e Tecnologia de Coimbra. 1

HTTP Hyper Text Transfer Protocol. 84–92, 94–96

HTTPS Hyper Text Transfer Protocol Secure. 8

JSON JavaScript Object Notation. 72, 92

JVM Java Virtual Machine. 38

JWT JSON Web Token. 58

LDAP Lightweight Directory Access Protocol. 36

REST Representational State Transfer. 52

SaaS Software as a Service. 54

SGBD Sistemas de Gestão de Base de Dados. 35, 52, 66, 67, 74, 76, 77, 79, 80, 82, 86, 95, 103

SPA Single Page Application. 37

SSO Single sign-on. 36

SVE Sistema de Votação Eletrónica. 13, 14

SVER Sistema de Votação Eletrónica Remota. 13, 14

SVU Sistema de Votação em Urna. 6

TLS Transport Layer Security. 64

UUID Universally unique identifier. 89, 90, 92

vCPUs Virtual CPU. 98

VPC Virtual Private Cloud. 3, 98

VST Voter Security Token. 39, 40, 43, 106

WAF Web Application Firewall. 60

XSS Cross-site scripting. 53, 117

Esta página foi deixada em branco propositadamente.

Lista de Figuras

3.1 Exemplo de uma mix network 20
3.2 Exemplo do funcionamento da criptografia homomórfica 21
3.3 Comparação entre criptografia não homomórfica e homomórfica 21
3.4 Diagrama sequência funcionamento geral das *Blind Signatures* 25
3.5 Entrada caverna do Ali Baba 27
3.6 Entrada de Alice pelo caminho B na caverna do Ali Baba 27

4.1 Boletim de voto de exemplo 31
4.2 Tabelas **P**, **Q**, **R** e **S** 33
4.3 Tabelas **P**, **Q**, **R** e **S** após a eleição 34
4.4 Seleção de candidato e verificação do recibo de voto 41
4.5 Comparação do EVIV com outros sistemas de votação eletrónica remota . . 42

6.1 Arquiteturas *single* e *multi tenant* 55
6.2 Exemplo de autenticação baseada em sessões 57
6.3 Exemplo de autenticação baseada em *tokens* 58
6.4 Exemplo de autenticação por chave de *API* 59
6.5 Arquitetura proposta 60
6.6 Diagrama sequência generalizado das interações Utilizador, Servidor orçamento participativo e o Servidor eleitoral 61
6.7 Diagrama de casos de uso de interações dos atores com o sistema 62

7.1 Diagrama sequência: Criação de eleição 66
7.2 Diagrama sequência: Edição de eleição 67
7.3 Diagrama sequência: Registo de eleitor 69
7.4 Diagrama sequência: Registo de *trustee* 70
7.5 Diagrama sequência: Eliminação de um *trustee* 71
7.6 Diagrama sequência: Congelamento de uma Eleição 73
7.7 Diagrama sequência: Adição de um voto encriptado 75
7.8 Diagrama sequência: Obter detalhe de um voto 76
7.9 Diagrama sequência: Contabilizar os votos 78
7.10 Diagrama sequência: Submeter os *decryption factors* e as *decryption proofs* dos *trustees* 79
7.11 Diagrama sequência: Combinar os decryption factors dos *trustees* 81
7.12 Diagrama sequência: Submissão de voto auditado 82

8.1 Criação da chave de API para execução dos testes 85
8.2 Evolução dos tempos resposta no congelamento de eleições 101
8.3 Evolução dos tempos de resposta no registo de voto 103
8.4 Alterações para otimização do processo de contabilização dos votos de uma eleição 104

Esta página foi deixada em branco propositadamente.

Lista de Tabelas

2.1 Exemplo de boletim de voto - Seleção de um único candidato 15
2.2 Exemplo de boletim de voto - Seleção de múltiplos candidatos 15
2.3 Exemplo de boletim de voto - Seleção de múltiplos candidatos com pontos . 16
2.4 Exemplo de boletim de voto - Referendo 16
2.5 Exemplo de boletim de voto - Votação preferencial 16
2.6 Exemplo resultado votação com três rondas 16
2.7 Exemplo resultado votação com duas rondas 17

4.1 Code Card exemplo 40

5.1 Orçamentos participativos por regra 47

6.2 Requisitos funcionais 52

8.1 Tempos de resposta na criação de N eleições em 30 segundos 99
8.2 Tempos de resposta no registo de N eleitores em 30 segundos 100
8.3 Tempos de resposta no congelamento de eleições com 1000, 10K e 20K eleitores 101
8.4 Tempos de resposta no congelamento de eleições de 30K, 40K e 50K eleitores 101
8.5 Tempos de resposta na adição de votos encriptados numa eleição 102
8.6 Tempos de resposta e consumo de memória na contabilização dos votos . . . 104
8.7 Tempos de resposta e consumo de memória na contabilização dos votos após otimização 104

Esta página foi deixada em branco propositadamente.

Capítulo 1

Introdução

Desde o início dos estados democráticos que o **direito de voto** se assume como um **pilar fundamental**. Ao longo da história observaram-se várias formas de voto, através da voz, do papel, e mais recentemente através de meios eletrónicos e tecnológicos. Nos países democráticos o exercício do voto é uma das mais importantes contribuições que os cidadãos podem conceder, elegendo os candidatos, sejam eles pessoas ou não, que acreditam servir melhor um dado propósito. O voto é tão poderoso que levou o célebre Abraham Lincoln, 16º presidente dos Estados Unidos da América, a afirmar que "**O voto é mais poderoso que uma bala**" numa referência facto de um voto poder trazer mais mudanças do que uma bala.

A dissertação que aqui se apresenta é realizada no âmbito do estágio do Mestrado em Segurança Informática e pretende apresentar uma proposta para votação eletrónica remota que, sob o escrutínio da comunidade, consiga garantir um elevado grau de confiabilidade no cumprimento dos principais requisitos e que se adeqúe a votações em orçamentos participativos. Adicionalmente, pretende-se proceder à implementação de uma *RESTful API*, baseada na proposta escolhida, que possa ser integrada com as atuais implementações das plataformas de orçamento participativo detidas pela entidade acolhedora. A elaboração desta dissertação foi realizada durante o ano letivo de 2019/2020, sob a orientação do Professor Doutor Fernando Boavida da Faculdade de Ciências e Tecnologia de Coimbra (FCTUC) e do Engenheiro Pedro Agante da Libertrium.

Os orçamentos participativos são mecanismos de democracia participativa, que dão o poder de decidir ou influenciar decisões de como e onde serão aplicadas parte das verbas dos orçamentos públicos. Estes mecanismos apresentam, cada vez mais, uma maior relevância, estando já a ser promovidos pelos continentes Europeu, Americano, Asiático, Africano e, por fim, na Oceania. Em Portugal, em concreto, muitos municípios e até governos promovem já este mecanismo tão importante. Cidades como Cascais, Lisboa, Faro, Lagoa (Algarve e Açores), Coimbra e Viseu são apenas alguns exemplos onde este mecanismo é aplicado no país.

Tendo em conta a importância destes mecanismos, torna-se imperativo garantir que estes fornecem um nível adequado de segurança, integridade, privacidade, confidencialidade e confiabilidade. De facto, existem um conjunto de requisitos fundamentais que são a base dos processos eleitorais. Garantir o seu cumprimento é fundamental e necessário para se terem processos eleitorais legítimos nos quais os eleitores possam confiar. Verificando-se que a votação nos orçamentos participativos é realizada essencialmente através de meios eletrónicos, mais concretamente através de plataformas para a *web*, considerou-se que seria importante garantir o exercício do direito de voto da forma mais segura e confiável. Nesta

dissertação, procurar-se-á encontrar uma proposta que satisfaça os requisitos mais importantes sem, todavia, comprometer a rapidez e facilidade no exercício do voto por parte dos eleitores.

1.1 Motivação

Como referido, um pouco por todo o mundo, têm vindo a ser realizados vários orçamentos participativos. Em especial, os países democráticos têm levado avante estas iniciativas. Dada a conjuntura sócio-económica atual, os orçamentos participativos apresentam-se como uma mais valia para os cidadãos de um país, pois estes podem decidir ou influenciar decisões de como e onde serão aplicados os fundos do orçamento (governamental ou municipal). É da nossa opinião que estes são uma grande mais valia para todos e vários países têm percebido exatamente isso e promovido estas iniciativas juntos dos seus cidadãos.

Segundo consta no *Participatory Budgeting World Atlas de 2019*, no mundo, na altura da recolha dos dados, existiam entre 11690 a 11825 orçamentos participativos distribuídos pelo continente Americano, Europeu, Asiático, Africano, Oceania. Considerando os orçamentos participativos um importante elemento democrático e verificando-se que os meios eletrónicos remotos são cada vez mais utilizados para a participação nos mesmos, considerou-se que faria todo o sentido dotar as plataformas de orçamentos participativos, geridas pela entidade acolhedora, de uma solução, validada pela comunidade, que garanta o cumprimento dos requisitos fundamentais de processos de votação. Desta forma, os eleitores terão maiores garantias e confiança no que diz respeito às votações em orçamentos participativos.

Tendo em conta o referido, pretende-se prover a entidade acolhedora de uma solução robusta, validada e escrutinada pela comunidade, que possa ser integrada com as plataformas existentes. Neste sentido é importante não acrescentar processos e/ou interações de elevada complexidade para os eleitores, pois isso poderia fazer com que desistissem de votar. Adicionalmente, é essencial garantir que, em nenhum momento da eleição, é possível saber ou determinar o resultado da eleição sem que a mesma tenha terminado, independentemente do tipo de utilizador ou até mesmo do acesso à base de dados onde os votos são armazenados.

1.2 Objetivos da proposta

O objetivo do trabalho realizado no âmbito da presente dissertação é fundamentalmente recorrer a uma proposta, validada e escrutinada pela comunidade, que garanta os requisitos dos sistemas de votação eletrónicos remotos e implementar uma *RESTful API* baseada nessa mesma proposta. Os orçamentos participativos são um dos mecanismos cujo exercício do voto é essencialmente feito recorrendo a sistemas de votação eletrónicos remotos, em especial através de plataformas *web*.

Tendo em linha de conta que a entidade fornece plataformas de orçamentos participativos, foi lançado o desafio de se investigar, dentro das inúmeras propostas para votação eletrónica remota, uma proposta que, em primeiro lugar, permita votações de acordo com as várias regras que se verificam definidas nos orçamentos participativos promovidos em Portugal e, em segundo lugar, que garanta o cumprimento dos seguintes requisitos:

- Unicidade
- Anonimato
- Autenticidade
- Integridade
- Auditabilidade

Por fim, espera-se que o produto deste trabalho possa vir a ser globalmente utilizado nos orçamentos participativos que estão à responsabilidade da entidade acolhedora.

1.3 Contribuições

Tendo em conta todo o trabalho realizado, no contexto desta dissertação, as principais contribuições são as seguintes:

- Proposta de arquitetura com separação lógica dos vários módulos (plataforma orçamento participativo, servidores de autenticação, sistemas de gestão de bases de dados e servidores eleitorais).
- Proposta de um serviço para garantir proteção face a ataques de negação de serviço e também face a ataques específicos para aplicações web.
- No sentido de expor o menos possível o servidor eleitoral, mas não só, propôs-se ter uma rede privada, apenas acessível por servidores da mesma Virtual Private Cloud (VPC), e uma rede pública.
- Proposta de um sistema de votação eletrónico, de código aberto, que se assume bastante adequado para a votação nos orçamentos participativos.
- Implementação de uma *RESTful API multi tenant* para gestão de todo o processo eleitoral dos orçamentos participativos. Note-se, todavia, que outras plataformas podem também ser integradas desde que a forma de votação e regras sejam compatíveis.

1.4 Organização do documento

Este documento está organizado em 9 capítulos. No capítulo 1 introduzem-se o direito de voto, os orçamentos participativos e a sua importância, a motivação para a realização deste trabalho e os principais objetivos. No capítulo 2 são apresentados os sistemas de votação (tradicional em urna e eletrónica), os seus requisitos e a forma como se procura garantir o seu cumprimento e também os diversos tipos de votação.

No capítulo 3 procedeu-se a uma breve análise de algumas das primitivas criptográficas mais frequentemente utilizadas no contexto de sistemas de votação eletrónica remota segura. Neste contexto, apresentar-se-ão as *Mix networks*, a criptografia homomórfica, as *Blind Signatures* e, para finalizar o capítulo, as *Zero-Knowledge Proofs*.

No capítulo 4 é feita uma breve análise de três sistemas de votação eletrónica. O primeiro, Scanintegrity II, um sistema de votação eletrónica presencial utilizado em eleições em alguns estados dos Estados Unidos da América. O segundo, um sistema de votação

eletrónica remota de código aberto (Helios Voting) utilizado em várias eleições cuja coercibilidade não apresente um risco considerável. Por fim, apresenta-se o EVIV, uma proposta de investigadores portugueses para votação eletrónica segura, que considera, inclusive, o problema de votação através de terminais inseguros.

No capítulo 5 introduzem-se os orçamentos participativos. Apresentam-se as fases dos orçamentos participativos, os sistemas de votação utilizados, o funcionamento do registo dos eleitores no caderno eleitoral, as regras de votação e, por fim, o funcionamento do processo de votação.

No capítulo 6 começa-se por especificar a solução a desenvolver, iniciando-se com a análise de requisitos (funcionais e não funcionais). Ainda neste capítulo faz-se um enquadramento da solução, apresentando os principais riscos aplicacionais verificados em aplicações para a web (na qual a solução proposta se enquadra), abordando-se também as arquiteturas *single* e *multi tenant*, a disponibilidade de serviço, alguns métodos de autenticação que podem ser utilizados na solução, a arquitetura base proposta, os atores do sistema, a *framework*, a linguagem de programação e também a biblioteca Helios.

No capítulo 7 aborda-se a implementação da *RESTful API*. Neste contexto, apresenta-se o método de autenticação escolhido e as principais alterações dos algoritmos de *hashing* utilizados pela biblioteca Helios. Por fim, procede-se à descrição da implementação de cada um dos requisitos funcionais identificados.

No capítulo 8 procede-se à validação da proposta implementada, começando-se por descrever alguns dos testes funcionais escritos para as várias funcionalidades. Por fim, apresentam-se também os testes não funcionais que foram realizados, com recurso a um serviço, as métricas obtidas, as otimizações necessárias e comparação entre a versão otimizada e a versão não otimizada. O capítulo 9, conclui esta dissertação analisando todo o trabalho realizado, os resultados alcançados e o trabalho futuro.

Esta página foi deixada em branco propositadamente.

Capítulo 2

Sistemas de votação

O direito de voto assume-se atualmente como um dos direitos mais importantes. De facto, é considerado um dos pilares fundamentais de estados democráticos. Atualmente, este direito é utilizado em situações distintas desde, por exemplo, uma simples eleição de delegados de turma até a actos de uma grande importância, como a eleição do presidente de um país.

Ao longo da história foram vários os sistemas desenvolvidos para o exercício do direito de voto. Cada um destes sistemas deve garantir um conjunto de requisitos. Neste capítulo pretende-se dar a conhecer esses requisitos, conhecer a evolução dos sistemas de votação, os seus principais problemas e também as suas vantagens.

2.1 Requisitos Gerais Votação Tradicional em Urna

A análise da literatura [7, 8, 15, 21] revela a existência de um conjunto de requisitos fundamentais para que uma votação se possa considerar válida e credível. De seguida, apresentam-se alguns dos mais importantes e referenciados requisitos que os Sistemas de votação em urna (SVUs) devem satisfazer.

- **Unicidade** - um eleitor pode votar apenas uma vez.

Como é este requisito garantido? O requisito é garantido através do registo da presença do eleitor na votação. Este registo é, por norma, efetuado em vários cadernos e por vários responsáveis pela mesa de voto. Desta forma, diminui-se a probabilidade de erro, mas também de fraude, por parte de algum dos membros da mesa. Note-se, no entanto, que se existir um conluio por parte dos vários representantes da mesa, esta diminuição pode não se verificar.

- **Anonimato** - não é possível associar um voto a um determinado eleitor.

Como é este requisito garantido? Através da utilização de boletins de voto sem qualquer identificador, que permita o relacionamento com o eleitor, juntamente com o facto de o preenchimento do boletim ser feito numa câmara de voto (onde apenas o eleitor se encontra) e pela urna lacrada onde são depositados os boletins de todos os eleitores.

- **Autenticidade** - apenas pessoas autorizadas podem votar.

Como é este requisito garantido? Os indivíduos que reúnem condições para o exercício de voto são registados nos cadernos eleitorais. Este processo ocorre, por norma, antes do dia ou dias designados para a votação. Quando um indivíduo se apresenta junto da mesa de voto faculta a sua identificação, que é validada, sendo depois verificado se reúne condições para poder exercer o voto (geralmente através dos cadernos eleitorais).

- **Integridade** - os votos dos eleitores não são modificados ou eliminados durante nem após o período eleitoral.

Como é este requisito garantido? Este requisito prevê-se ser satisfeito através da urna fechada durante o processo eleitoral. Após este período, no momento da contabilização dos votos, a presença de elementos representantes das várias partes a votação pretende garantir que não existe qualquer manipulação da votação. Note-se, no entanto, que existindo conluio entre os representantes este requisito pode ser facilmente quebrado.

- **Auditibilidade** - é necessário conseguir-se verificar que todo o processo de votação correu de acordo com o previsto e que todos os votos foram contabilizados corretamente.

Como é este requisito garantido? Os boletins de voto físicos são fundamentais para permitirem a verificação do processo eleitoral. As autoridade eleitorais, se assim o acharem pertinente ou necessário, podem, a qualquer momento, confrontar os registos físicos dos cadernos eleitorais e boletins de voto, de forma a perceber se os resultados reportados correspondem, de facto, à verdade.

Mais uma vez, este processo é acompanhado por representantes das várias partes a votação. Desta forma, a probabilidade de manipulação da votação é diminuta.

- **Não-coercibilidade** - não deve ser possível aos eleitores provarem em quem é que votaram. A garantia de satisfação deste requisito é muito importante pois, caso a mesma não seja satisfeita, é então possível a coerção e/ou venda de votos.

Como é este requisito garantido? Este requisito pretende ser garantido através da utilização da câmara de voto onde apenas consta fisicamente o eleitor. Não existindo mais ninguém a testemunhar o exercício do voto do eleitor, este requisito é satisfeito.

Note-se, no entanto, que a total garantia da satisfação deste requisito torna-se difícil nos dias atuais. Um eleitor pode, facilmente, ser alvo de coerção em que lhe seja exigido que leve com ele algum meio de transmissão de vídeo para que possa comprovar que realizou o processo conforme as instruções. A tecnologia atual pode passar facilmente despercebida e comprometer este requisito. Todavia, consideramos que é altamente improvável coagir um número suficiente de pessoas para haver implicações no processo eleitoral. Da mesma forma, seria difícil que essa ação coerciva não fosse notada, ou pela supervisão ou pela confissão de algum envolvido, pelas entidades eleitorais.

2.2 Requisitos Gerais Votação Eletrónica

À semelhança da votação em urna a votação eletrónica, para ser considerada válida e credível, deve satisfazer um conjunto de requisitos.

Através da revisão de literatura, obteve-se um conjunto de requisitos [7, 8, 15, 21], que devem ser garantidos. De seguida, apresentam-se aqueles que, na literatura analisada, foram mencionados mais frequentemente como relevantes.

- **Unicidade** - um eleitor pode votar apenas uma vez.

Como é este requisito garantido na votação eletrónica presencial? O requisito é garantido através do registo da presença do eleitor na votação. Este registo é, por norma, efetuado em vários cadernos (eletrónicos ou não) e por vários responsáveis pela mesa de voto. Desta forma diminui-se a probabilidade de erro mas também de fraude por parte de algum dos membros da mesa.

Como é este requisito garantido na votação eletrónica remota? Na votação eletrónica remota tenta-se garantir o cumprimento deste requisito através da implementação de mecanismos de autenticação do utilizador para que se possa verificar se: 1º - o utilizador reúne condições para exercer o direito de voto; 2º - o utilizador ainda não realizou o seu exercício de voto no processo eleitoral. A implementação do sistema deve equacionar este requisito garantindo também que não é possível relacionar um voto com um utilizador, garantindo assim um dos requisitos basilares: **o anonimato**.

- **Anonimato** - não é possível associar um voto a um determinado eleitor.

Como é este requisito garantido na votação eletrónica presencial? Este é um dos requisitos mais importantes e críticos de um processo eleitoral. A garantia deste requisito é o que permite um voto livre sem coação. Na **votação em urna**, como vimos anteriormente, este requisito é garantido através da presença do eleitor na câmara de voto (onde apenas este se encontra) e pela colocação do boletim de voto (sem qualquer identificador) na urna.

Na votação eletrónica presencial, apesar do exercício do voto continuar a ser realizado em câmara de votação, devidamente isolada, para que este requisito continue a ser cumprido, é necessário que não exista qualquer associação entre a autenticação do eleitor e o voto. Se o processo de autenticação for o tradicional (processo descrito anteriormente), e desde que não exista qualquer registo da data/hora em que o eleitor se apresentou para votar, então não existe qualquer associação estando o anonimato garantido. Se, por outro lado, o processo de autenticação for realizado recorrendo a meios eletrónicos, torna-se vital que não exista qualquer associação possível entre a autenticação e o voto que um eleitor realizou.

Como é este requisito garantido na votação eletrónica remota? Como vimos, na votação eletrónica presencial existem já alguns problemas que podem colocar em causa este requisito. Na votação remota esta situação torna-se crítica. Para além dos problemas referidos para a votação eletrónica presencial, a votação eletrónica remota enfrenta ainda mais algumas lacunas que fazem com que este requisito seja difícil de garantir. Alguns exemplos dessas lacunas têm que ver com:

1. **Canal de comunicação** - Um dos canais mais utilizados na votação eletrónica remota é a internet. Ainda que sistemas desta natureza recorram geralmente a protocolos seguros, como o Hyper Text Transfer Protocol Secure (HTTPS) [1], para a transmissão dos dados, existe sempre a possibilidade de um atacante interceptar as comunicações e tentar desencriptar a comunicação. Se o mesmo for bem sucedido, então este requisito pode ser comprometido.

[1] Extensão segura do protocolo HTTP.

2. **Meios de votação** - O exercício do voto através de meios eletrónicos remotos é realizado através de dispositivos considerados inseguros. Computadores, *tablets* e *smartphones* são exemplos destes dispositivos. Muitos deles são facilmente infetados com *malware* [2] que pode facilmente comprometer o anonimato.

- **Autenticidade** - apenas pessoas autorizadas podem votar.

Como é este requisito garantido na votação eletrónica presencial? A autenticação do indivíduo na votação eletrónica presencial é muito semelhante ao processo realizado na votação em urna. De facto, algumas arquiteturas têm descartado esta fase mantendo a mesma tal e qual o processo tradicional em urna [8]. É, no entanto, expectável que este processo possa também passar a ser suportado por meios eletrónicos.

A utilização de equipamentos biométricos (irís, impressão digital, ou voz), em conjunto com outros métodos de autenticação, podem vir a ser utilizados para a identificação inequívoca de um indivíduo [8]. Recorrer à utilização da impressão digital parece a solução mais viável, pois, em Portugal, o documento de identificação já possui estes dados. Todavia, existe uma forte controvérsia em relação à utilização destes dados que nos caracterizam e identificam inequivocamente, para os referidos fins.

Como é este requisito garantido na votação eletrónica remota? A autenticação do indivíduo na votação eletrónica remota traz novos desafios. Métodos de autenticação mais tradicionais, como utilizador/email e password, poderão não ser suficientes. Torna-se, portanto, necessário que os mecanismos e arquiteturas implementadas em sistemas de votação eletrónica remota, sejam capazes de identificar o indivíduo, idealmente recorrendo a mais que uma forma de autenticação, de forma segura.

Note-se que a implementação de mecanismos e arquiteturas devem ter em conta a criticidade/importância do processo e as ameaças que se lhe apresentam. Eleições europeias, autárquicas e nacionais apresentam um risco muito elevado quando comparadas a votações, por exemplo, em orçamentos participativos.

- **Integridade** - os votos dos eleitores não são modificados ou eliminados, nem durante nem após o período eleitoral.

Como é este requisito garantido na votação eletrónica presencial e remota? A integridade é um requisito muito importante. Sem o cumprimento deste requisito não existe credibilidade no processo. Nos sistemas de votação eletrónica os votos são armazenados digitalmente, em "urnas eletrónicas", para depois serem contabilizados. Na tentativa de se garantir a integridade dos votos torna-se necessário recorrer a canais de comunicação seguros, à utilização de algoritmos criptográficos seguros, abertos e sob um forte escrutínio por parte da comunidade criptográfica.

A utilização de tecnologias como a Blockchain poderão ajudar a mitigar este problema pelas suas características próprias. Existem atualmente várias propostas de implementações de sistemas de votação electrónica suportados na Blockchain [14, 22, 27, 28]. Características como a imutabilidade são atrativos para a implementação destes sistemas na Blockchain. No entanto, esta tecnologia não é perfeita e, como tal, existem algumas desvantagens que devem ser consideradas.

[2] Software malicioso destinado a infiltrar-se, de forma ilícita, num sistema para, por exemplo, causar dano, manipular ações ou extrair informação do mesmo.

- **Auditibilidade** - é necessário conseguir-se verificar que todo o processo de votação correu de acordo com o previsto e que todos os votos foram contabilizados corretamente.

Como é este requisito garantido na votação eletrónica presencial? Na votação eletrónica presencial os votos são geralmente armazenados em urnas eletrónicas. Se forem somente armazenados digitalmente, então um auditor apenas poderá basear-se nas evidências digitais que existem, que podem ser apagadas ou sofrer de alguma anomalia, para poder avaliar a conformidade do processo de votação.

A dependência de meios exclusivamente digitais apresenta alguns problemas e, por isso, várias implementações recorrem a impressoras para produzir uma evidência física, que depois é colocada em urna tradicional. Desta forma os auditores têm uma outra fonte, materializada, de informação, que poderão usar para perceber se, de facto, o processo e o sistema de votação estão a funcionar como previsto. O registo de determinados eventos, desde que os mesmos não coloquem em causa os restantes requisitos, deve também ser implementado para que, caso seja necessário, estes possam ser consultados e se possa extrair informação que possa ajudar no processo de auditoria.

Como é este requisito garantido na votação eletrónica remota? Na votação eletrónica remota, como os eleitores não se dirigem a um local físico para exercer o direito de voto, não existe a possibilidade de os mesmos colocarem uma evidência física numa urna tradicional. Uma vez que não existe qualquer registo materializado, então um auditor apenas poderá basear-se nas evidências digitais para poder fazer a sua análise. No entanto, como vimos, as evidências digitais podem também ser eliminadas deixando um rasto muito ténue que pode não permitir a um auditor aferir quaisquer conclusões.

Como é que pode então um eleitor ter certezas de que um sistema de votação eletrónico é seguro e confiável? Em primeiro lugar, estes sistemas deverão ser certificados por uma entidade, idónea, que comprove que o sistema foi implementado utilizando mecanismos e algoritmos criptográficos comprovadamente seguros, e que foram aplicadas as melhores práticas de desenvolvimento seguro durante todo o ciclo de desenvolvimento de *software*.

Até ao momento apenas falámos em auditoria "interna", isto é, auditoria levada a cabo por uma entidade contratada, que seja idónea, e que verifique o correto funcionamento e implementação do sistema. Uma questão que se pode impor é se esta auditoria não poderá ser realizada externamente, por outros moldes, uma vez que não existe um acesso aberto ao sistema e ao seu código, até mesmo pelos próprios cidadãos. Algumas implementações de sistemas prevêem algo que denominam de End to End Verifiable (E2E-V). Implementações E2E-V permitem aos eleitores verificar que o sistema de votação recebeu o seu voto, interpretou-o corretamente e o utilizou no processo de contabilização de votos como pretendido.

- **Não-coercibilidade** - não deve ser possível aos eleitores provarem em quem é que votaram. A garantia de satisfação deste requisito é muito importante pois, caso a mesma não seja satisfeita, é então possível a coerção e/ou venda de votos.

Como é este requisito garantido na votação eletrónica presencial? Anteriormente, na votação tradicional em urna, vimos como é que se procurava tentar garantir a satisfação deste requisito e quais os problemas que o poderiam comprometer. Na votação eletrónica presencial, a abordagem utilizada para se tentar garantir a satifação do requisito é a mesma. Como tal, os problemas que se apresentam a este sistema de votação são os mesmos que foram identificados anteriormente.

Como é este requisito garantido na votação eletrónica remota? Os sistemas de votação presencial, quer o tradicional quer o eletrónico, apresentam alguns problemas na garantia de satisfação do requisito de não-coercibilidade. No sistema eletrónico remoto este requisito torna-se ainda mais crítico e complexo de satisfazer, uma vez que a votação pode ser feita à vista de outras pessoas que podem assim comprovar o sentido de voto [8]. De facto, a satisfação deste requisito em sistemas de votação eletrónica remota será muito complicada, ou até mesmo impossível, de se conseguir atingir.

- **Disponibilidade** - O sistema deve estar disponível aos eleitores durante todo o período eleitoral

Como é este requisito garantido na votação eletrónica presencial?

Garantir a satisfação deste requisito é muito importante para a não negação do direito de voto de um eleitor. Uma das formas de se conseguir garantir o cumprimento deste requisito é através de redundância dos equipamentos e sistemas. Desta forma, se algum desses falhar, existe sempre um outro que garantirá a disponibilidade. Todavia, na votação presencial pode sempre existir a opção de votação tradicional em urna, como parte de um plano de contingência, na eventualidade de não ser mesmo possível garantir o direito de voto através dos sistemas eletrónicos.

Como é este requisito garantido na votação eletrónica remota?

Na votação eletrónica remota, garantir a disponibilidade de um sistema de votação torna-se uma tarefa ainda mais importante e complexa. Geralmente, este tipo de sistemas utilizam a internet como meio de transmissão de dados. A exposição destes sistemas nesta rede abre porta a potenciais ataques à disponibilidade de serviço. Um ataque bem sucedido pode negar o direito de voto a eleitores legítimos, causando impactos no processo eleitoral.

De forma a detetar e mitigar este tipo de ataques, é necessário uma infraestrutura de elevada capacidade, com monitorização continua, e especializada na deteção e mitigação deste tipo de ataques. Nos dias atuais existem serviços especializados, com infraestruturas de elevada capacidade e "inteligência", para detetar e mitigar estes ataques. Um dos serviços mais conhecidos e que pode ser facilmente integrado é fornecido pela CloudFlare.

2.3 Sistemas de Votação Tradicionais

Uma das formas de votação mais tradicionais, amplamente utilizada por vários países nas eleições, é a votação em papel, através de boletins de voto, que depois são depositados em urnas, seladas, através de uma pequena ranhura[7, 8]. Este é o sistema de votação utilizado em países como, por exemplo, Portugal, Japão e Alemanha. Note-se que, apesar do sistema utilizado ser o mesmo em vários países, podem existir especificidades neste processo.

O sistema de votação descrito funciona, geralmente, em quatro fases descritas de seguida [8, 15, 36]:

1 - Recenseamento

O recenseamento consiste no registo de um cidadão como elegível para um processo eleitoral. Entidades competentes são responsáveis por registar os cidadãos que reúnem condições para exercer o voto [8, 36, 37]. Atualmente, em Portugal, este registo é feito de forma automática quando o cidadão passa a reunir as condições exigidas por lei para poder exercer o seu direito de voto.

2 - Validação da identidade do eleitor

Nesta fase, as autoridades eleitorais procedem à validação da identidade do cidadão, analisando por exemplo o documento identificativo, e verificam se o mesmo consta no caderno eleitoral [8, 15, 36, 37]. Caso o cidadão reúna as condições necessárias para poder exercer o direito de voto, é-lhe dado o boletim de voto e a sua presença é registada, evitando, assim, que vote mais que uma vez.

Nota: Caso o cidadão não esteja no caderno eleitoral, não poderá exercer o voto e o processo termina nesta fase.

3 - Recolha do voto

É nesta fase que o cidadão eleitor exerce, de facto, o seu voto preenchendo o boletim, de acordo com as regras. Note-se que estas regras podem diferir de país para país ou até mesmo de acordo com o tipo de eleição.

Após o preenchimento do boletim de voto, o cidadão dobra o seu boletim, para que a sua escolha não seja visível, e insere o mesmo na urna, finalizando, assim, o seu direito de voto [8, 15, 36, 37].

4 - Contabilização dos votos

Após o término do período de votação, os funcionários eleitorais procedem à abertura das urnas e começam a classificar (voto em branco, nulo ou válido) e a contabilizar os votos [8, 15, 36, 37]. Após a análise de todos os boletins contidos nas urnas, os resultados são, então, comunicados.

Apresentam-se de seguida as principais vantagens e desvantagens deste sistema.

Principais vantagens [8, 36, 37]:

- Compreendida facilmente pelos eleitores
- Maior transparência. Os eleitores percebem facilmente como e onde os votos são armazenados e de que forma é garantido o anonimato do voto.
- Facilmente auditável. Existem registos físicos do voto (boletins de voto e cadernos eleitorais com as presenças).
- Sistema mais utilizado e confiado até ao momento, especialmente em votações com um número de eleitores baixo.

Principais desvantagens [8, 36, 37]:

- Tempo elevado para o apuramento dos resultados.
- Suscetível a erros de contabilização.
- Suscetível a adulterações.

2.4 Sistemas Eletrónicos Presenciais

Um Sistema de Votação Eletrónica (SVE) consiste na utilização, tal como o próprio nome o sugere, de meios eletrónicos no processo da votação. Algumas das principais vantagens da utilização destes sistemas têm que ver com a diminuição de custos, agilização do processo e rapidez na contabilização dos votos. No entanto, estes sistemas introduzem também alguns problemas que podem comprometer a segurança, privacidade e/ou a integridade do processo, que abordaremos posteriormente.

Este sistema é muito semelhante ao sistema de votação tradicional. De facto, os processos e fases são exatamente os mesmos, mas recorrendo a meios digitais para a validação/identificação do eleitor e para o seu exercício do direito de voto. Também neste sistema continua a existir a necessidade de o eleitor se dirigir a um local, definido previamente pelas autoridades eleitorais, para exercer o seu voto. Salienta-se que apesar de o processo recorrer a meios digitais, existem pessoas responsáveis, à semelhança do sistema tradicional, que acompanham todo processo.

A investigação em técnicas, sistemas e arquitecturas para votação eletrónica tem sido objeto de interesse de governos e investigadores um pouco por todo o mundo. As várias contribuições têm tentado dar resposta aos principais perigos que os SVEs apresentam, de forma a que estas votações possam apresentar-se como uma alternativa segura e viável. No entanto, até ao momento, não foi encontrada nenhuma solução "perfeita". Apesar disso, as propostas que têm vindo a ser sugeridas, trazem contribuições que permitem tornar os SVE mais seguros e confiáveis recorrendo, essencialmente, a mecanismos, técnicas e algoritmos criptográficos.

Apresentam-se de seguida as principais vantagens e desvantagens deste sistema.

Principais vantagens [8, 15, 36, 37]:

- Apuramento de resultados mais rápido.
- Maior acessibilidade. É possível dotar os equipamentos de votos com *hardware* e *software* que permita o voto por pessoas com deficiência dando, consequentemente, uma maior autonomia.

Principais desvantagens [8, 15, 36, 37]:

- Menor transparência. Os eleitores não compreendem o funcionamento do sistema, nem como são armazenados os votos, causando desconfiança perante a votação.
- Sem possibilidade de auditoria/verificação por parte do eleitor, exceto nos sistemas E2E-V.
- Exposição a variadas ameaças digitais que podem comprometer a segurança do sistema.
- Obscuridade no processo.

2.5 Sistemas Eletrónicos Remotos

Um Sistema de Votação Eletrónica Remota (SVER) consiste na utilização de meios eletrónicos no processo de votação. Ao contrário do SVE, este sistema permite o exercício

do direito de voto remoto, quer isto dizer, a partir de qualquer lugar em que o eleitor se encontre. Nos dias atuais, grande parte dos cidadãos, nos países desenvolvidos, transporta consigo um *smartphone*, *tablet* ou computador com acesso à internet. As capacidades destes dispositivos permitem o exercício do direito ao voto de forma rápida, prática, fácil, intuitiva e acessível.

À primeira vista poder-se-á pensar que o SVER é o que mais se ajusta à sociedade atual que tem vindo a ficar cada vez mais tecnológica. No entanto, os SVE apresentam grandes desafios. Alguns desafios prendem-se com o facto de os dispositivos dos cidadãos não serem seguros, podendo ser infetados e comprometer os votos feitos através desses dispositivos. Outro problema são os ataques *Denial-of-service (DOS)* e *Distributed Denial-of-service (DDOS)* [15], que podem ser levados a cabo de forma a impedir a votação por parte dos eleitores. Um outro problema tem que ver com a desconfiança que os eleitores têm neste tipo de sistemas para eleições de carácter importante como as Europeias e Legislativas [8].

Este sistema apresenta, para os eleitores, algumas vantagens significativas. Uma dessas vantagens é a mobilidade. Os eleitores podem, a partir de um qualquer lugar, utilizar meios eletrónicos que lhes permitam, de forma rápida e fácil, exercer o seu direito de voto. Em votações eleitorais, onde os cidadãos de um país são chamados a escolher os seus representantes, e estando estes em várias partes do mundo, este sistema de votação torna-se ainda mais relevante e poderia até reduzir a abstenção [36].

Principais vantagens [7, 8, 15, 36, 37]:

- Apuramento de resultados mais rápido.
- Maior acessibilidade e autonomia, especialmente para pessoas com deficiência. Atualmente os equipamentos eletrónicos trazem várias opções de acessibilidade que podem ser utilizadas para facilitar o voto, de forma autónoma, a pessoas com deficiência.
- Maior mobilidade. Eleitores podem votar a partir de qualquer lugar. Eleitores que se encontrem no estrangeiro podem facilmente e comodamente exercer o seu direito de voto.
- Redução de custos, a médio/longo prazo. Eliminação dos boletins de voto em papel e redução do número de pessoas destacadas para o processo eleitoral trazem, consequentemente, uma redução significativa dos custos.

Principais desvantagens [7, 8, 36, 37]:

- Menor transparência. À semelhança do que acontece na votação eletrónica os eleitores não compreendem o funcionamento do sistema.
- Maior exposição a ameaças. Equipamentos dos eleitores podem estar comprometidos e colocar em causa o processo eleitoral.
- Sem possibilidade de auditoria/verificação por parte do eleitor, exceto nos sistemas E2E-V.
- Obscuridade no processo. O processo eleitoral é visto como uma caixa negra.
- Possibilidade de coercibilidade elevada.
- Fracas garantias sobre a confirmação do voto. O eleitor tem de ser capaz de verificar se o seu voto foi recebido pelo sistema, se foi interpretado corretamente, e se o seu voto foi *cast-as-intended*. Apenas os sistemas E2E-V permitem esta mesma verificação por parte do eleitor.

- Sem registo físico do voto.
- Aumento da suscetibilidade para atividades fraudulentas. Um atacante pode, por exemplo, fazer com que um voto em "A" seja de facto um voto em "B" sem que este se aperceba.

2.6 Tipos de votação

Ao longo da história tem-se verificado a necessidade de utilização de vários tipos de votação. A construção de um sistema de votação, que garanta a satifação dos requisitos fundamentais inerentes, é influenciada pelos tipos de votação que devem ser suportados por esse sistema. Nesta secção pretende-se introduzir alguns tipos de votação mais frequentemente utilizados.

Seleção de um único candidato - Como o próprio nome indica, neste tipo de votações, os eleitores devem seleccionar no máximo um candidato ou, em alternativa, deixar o boletim de voto em branco. A escolha de mais do que um candidato, quando possível, torna o voto nulo. No término da eleição, o candidato vencedor é apurado verificando-se qual é que possui o maior número de votos. Um exemplo pode ser visto na tabela 2.1.

Candidato	
Alice	X
Bob	
Carl	

Tabela 2.1: Exemplo de boletim de voto - Seleção de um único candidato

Seleção de múltiplos candidatos - Neste tipo de eleições, os eleitores têm um número mínimo e/ou máximo de candidatos em que podem votar. Tal como no exemplo anterior o vencedor é aquele que apresenta uma maior número de votos. Um exemplo, de uma votação com o máximo de 2 votos, pode ser visto na tabela 2.2.

Candidato	
Alice	X
Bob	
Carl	X

Tabela 2.2: Exemplo de boletim de voto - Seleção de múltiplos candidatos

Seleção de múltiplos candidatos com pontos - Dados X pontos os eleitores podem distribuir esses mesmos pontos pelos candidatos. Note-se que, poderão existir algumas regras neste tipo de votação. A título de exemplo poderá ser limitado o número de pontos máximo por candidato ou apenas limitar a atribuição da totalidade dos pontos a apenas um candidato. O apuramento do vencedor faz-se da mesma forma que nos tipos de votação anteriores. O(s) candidato(s) eleitos são aqueles que obtiverem o maior número de pontos. Um exemplo, numa votação com 3 votos de 1, 2 e 3 pontos pode ser visto na tabela 2.3.

Candidato	
Alice	2
Bob	3
Carl	1

Tabela 2.3: Exemplo de boletim de voto - Seleção de múltiplos candidatos com pontos

Referendo - Geralmente utilizados em assuntos políticos, os referendos são instrumentos de democracia direta, onde os eleitores são chamados a pronunciar-se, através do voto, sobre determinadas questões. Note-se também que os referendos podem ter carácter vinculativo ou não vinculativo. Por exemplo, na constituição da república portuguesa "o referendo só tem efeito vinculativo quando o número de votantes for superior a metade dos eleitores inscritos no recenseamento", como se pode ler no número 11 do artigo 115 da Constituição da República Portuguesa. Geralmente, neste tipo de votações, o voto responde a questões com "sim" ou "não". Um exemplo pode ser visto na tabela 2.4.

	Sim	Não
Concorda com...	X	

Tabela 2.4: Exemplo de boletim de voto - Referendo

Preferencial - Neste tipo de votação o eleitor atribui um valor, distinto, a cada candidato expressando a sua preferência. Por norma o apuramento dos candidatos eleitos faz-se através da determinação dos candidatos com maior percentual de preferência. Um exemplo pode ser encontrado na tabela 2.5.

Candidato	
Alice	1
Bob	4
Carl	3
Jane	2

Tabela 2.5: Exemplo de boletim de voto - Votação preferencial

Instant Runoff - Tipo de votação preferencial que permite a eleição de um único candidato através de um processo contínuo onde, a cada ronda, o candidato com a menor preferência é eliminado. Este processo repete-se tantas rondas quanto as necessárias até que haja um candidato que tenha mais de 50% dos votos. Um exemplo de boletim de voto pode ser visto na tabela 2.5 enquanto que um exemplo dos resultados de cada ronda, de uma votação com três rondas, pode ser encontrado na tabela 2.6.

Candidato	% Votos
Alice	15%
Bob	35%
Carl	27%
Jane	23%

Candidato	% Votos
Bob	50%
Carl	30%
Jane	20%

Candidato	% Votos
Bob (vencedor)	65%
Carl	35%

Tabela 2.6: Exemplo resultado votação com três rondas

Duas rondas - Neste tipo de votação existem, no máximo, duas rondas para eleição de um único candidato. Se um candidato tiver mais de 50% dos votos na primeira ronda este é o vencedor. Caso este requisito não seja verificado, os dois candidatos mais votados vão à segunda ronda de disputa e nesta, o que tiver mais de 50% dos votos vence. Ao contrário do que se verifica no *Instant Runoff* os eleitores apenas podem escolher um dos candidatos a votação. Um exemplo de boletim de voto pode ser visto na tabela 2.1, acrescida da Jane, enquanto que um exemplo dos resultados das rondas pode ser encontrado na tabela 2.7.

Candidato	% Votos
Alice	15%
Bob	35%
Carl	27%
Jane	23%

Candidato	% Votos
Bob	49%
Carl (vencedor)	51%

Tabela 2.7: Exemplo resultado votação com duas rondas

Contingent vote - Este é uma forma especial de *Instant Runoff* onde apenas existem duas rondas no máximo. Na primeira ronda são contabilizadas as primeiras preferências dos eleitores. Se algum dos candidatos detiver mais de 50% de preferência este é considerado imediatamente como vencedor. Caso nenhum dos candidatos detenha mais de 50% os dois candidatos com a maior preferência são transitados para uma segunda ronda. **Em primeiro lugar** todos os votos cuja primeira preferência não seja nenhum dos dois candidatos selecionados na primeira ronda, são transitados para esta segunda ronda. Depois, são contabilizadas as segundas preferências dos eleitores. O candidato com maior preferência nesta segunda ronda é o vencedor. Um exemplo de boletim de voto pode ser visto na tabela 2.5 enquanto que um exemplo dos resultados das rondas pode ser encontrado na tabela 2.7.

Exhaustive ballot - Este é mais um tipo de votação por múltiplas rondas que permite a eleição de um único candidato. Neste tipo de votação o eleitor marca no boletim de voto, no máximo, um candidato no qual deseja depositar o seu voto. Se na primeira ronda algum dos candidatos obtiver mais de 50% dos votos este é o vencedor da eleição. Caso nenhum dos candidatos detenha mais do que 50% dos votos é eliminado o candidato com o menor número de votos e é iniciada a próxima ronda. Este processo repete-se tantas rondas quanto as necessárias até que haja um candidato que tenha mais de 50% dos votos. Um exemplo de boletim de voto pode ser visto na tabela 2.1 enquanto que um exemplo dos resultados das rondas pode ser encontrado na tabela 2.6.

Este tipo de votação tem claras parecenças com a votação por Duas Rondas. Todavia, existem importantes diferenças. Enquanto que se na votação em Duas Rondas nenhum dos candidatos obtiver mais que 50% dos votos apenas os dois mais votados transitam para a segunda ronda onde será então apurado o vencedor, por outro lado, no *Exhaustive ballot* apenas um candidato, o menos votado, é eliminado por cada ronda.

Também podem ser encontradas algumas semelhanças com o *Instant Runoff*. Todavia, note-se, que o *Instant Runoff* é um tipo de votação preferencial no qual os eleitores atribuem uma preferência a cada um dos candidatos. Por outro lado, no *Exhaustive ballot* apenas é permitido ao eleitor votar, no máximo, em um dos candidatos.

2.7 Conclusão

No início do capítulo começou-se por definir os requisitos gerais para votação por sistemas tradicionais e eletrónicos. Desta análise resulta a conclusão de que os referidos sistemas partilham essencialmente dos mesmos requisitos. Todavia, para além dos meios de voto, muda a forma, os mecanismos e as técnicas utilizadas para garantir o cumprimento desses mesmos requisitos.

Procedeu-se de seguida à descrição das várias fases dos sistemas de votação tradicionais, nomeadamente através de boletim de voto físico, e funcionamento de cada uma destas. Neste contexto analisaram-se ainda as principais vantagens e desvantagens. Destes sistemas destacam-se, como **principais vantagens**, a **maturidade**, **facilidade de compreensão**, a **transparência** e **auditabilidade**. Por outro lado, a **desvantagem que mais se destaca** é o **tempo de apuramento mais elevado**, comparativamente à votação por meios eletrónicos.

Passando para os sistemas de votação eletrónicos presenciais, sistema cujo funcionamento é semelhante ao dos sistemas de votação tradicionais, analisaram-se as **principais vantagens** das quais se destacam a **diminuição do tempo de apuramento de resultados** e, em certos casos, a **maior acessibilidade** para pessoas com alguma deficiência. Por outro lado, como **principais desvantagens** destacam-se a **menor transparência**, a obscuridade do processo (para os eleitores) e a **exposição** a diversas **ameaças digitais**.

Para finalizar a análise dos sistemas de votação abordámos os sistemas eletrónicos remotos onde, depois de uma breve contextualização e referência de potenciais ameaças, se analisaram as suas principais vantagens e desvantagens. **Como vantagens** destacam-se a **mobilidade**, o **apuramento de resultados mais rápido**, **maior acessibilidade** e **autonomia**. Por outro lado, **como desvantagens**, destacam-se a **menor transparência** do processo (eleitor não percebe o funcionamento e como são garantidos os requisitos), **forte exposição a ataques digitais**, maior probabilidade de **coercibilidade** e, com exceção dos sistemas E2E-V, as **fracas garantias de confirmação do voto**.

Por fim, analisaram-se vários tipos de votação. Neste contexto, fizemos referência apenas a alguns dos mais conhecidos e utilizados. Começámos por abordar os tipos mais simples, como a **seleção de um único candidato**, passando para a **seleção de múltiplos candidatos**, com e sem pontos, e para o **referendo**. Após o referendo analisou-se a **votação preferencial**, onde um eleitor atribui uma preferência a cada um dos candidatos, o **Instant Runoff**, onde através de um processo contínuo e por rondas se irá eleger um único candidato, o **Contingent vote**, variação do Instant Runoff onde no máximo existem duas rondas e, por fim, o **Exhaustive ballot** sistema de múltiplas rondas (tantas quantas necessárias até que um dos candidatos obtenha mais de 50% dos votos) onde, a cada ronda, o candidato com menor percentagem de votos é eliminado.

Em suma, as análises feitas ao longo deste capítulo apresentaram-se fundamentais para conhecer os requisitos, a forma de funcionamento geral dos processos eleitorais, as vantagens e desvantagens de cada sistemas e a percepção dos diferentes tipos de votação que são geralmente utilizados no contexto de processos eleitorais.

Capítulo 3

Primitivas criptográficas

Num mundo cada vez mais dotado de meios tecnológicos, existe uma tendência natural para invocar a necessidade de poder exercer o direito de voto através destes mesmos meios. Em vários países, como a Suíça, Brasil e Canadá, a votação eletrónica em eleições governamentais é hoje uma realidade. No entanto, vários problemas são geralmente associados à sua utilização. Todavia, alguns investigadores têm tentado continuamente melhorar estes sistemas de forma a que garantam o cumprimento de todos os requisitos necessários e se tornem uma alternativa viável aos meios de votação tradicionais.

Nesta secção pretende-se dar a conhecer quatro primitivas criptográficas frequentemente utilizadas nas propostas de sistemas de votação eletrónica. São elas: *Mix networks*, criptografia homomórfica (*Homomorphic encryption*), *Blind Signatures* e as *Zero-Knowledge Proofs*.

3.1 Mix networks

As *mix networks*, vulgarmente conhecidas por *mixnets*, foram introduzidas em 1981 por David Chaum e, desde então, têm demonstrado ser uma ótima solução para problemas relacionados com o anonimato. São um tipo de protocolo de encaminhamento cujo principal objetivo é enviar mensagens através da internet, por exemplo, sem revelar a identidade/-localização do emissor da mensagem. Na proposta original [13], David Chaum, o objetivo era, através de criptografia assimétrica, permitir a troca de emails entre dois intervenientes de forma completamente anónima e de forma não rastreável. Todavia, estas têm vindo a ser utilizadas em sistemas de votação eletrónica para garantir que a ordem dos votos não é igual à ordem em que estes são realizados.

Neste protocolo, um conjunto de servidores denominados de servidores *proxy*, são utilizados para "esconder" a correspondência entre as entradas e as saídas (encriptadas). Cada servidor tem um par de chaves criptográficas, uma pública e outra privada, que lhes permite encriptar e desencriptar as entradas. De uma forma geral, e olhando para o caso concreto da utilização em sistemas de votação, um servidor *proxy* pode ser visto como uma caixa preta que aceita uma sequência de mensagens como entrada e retorna essas mensagens encriptadas e numa outra sequência. Através de um conjunto de permutações realizadas ao longo dos vários servidores, torna-se muito difícil descobrir a ordem original dos votos. Um exemplo pode ser visto na figura 3.1.

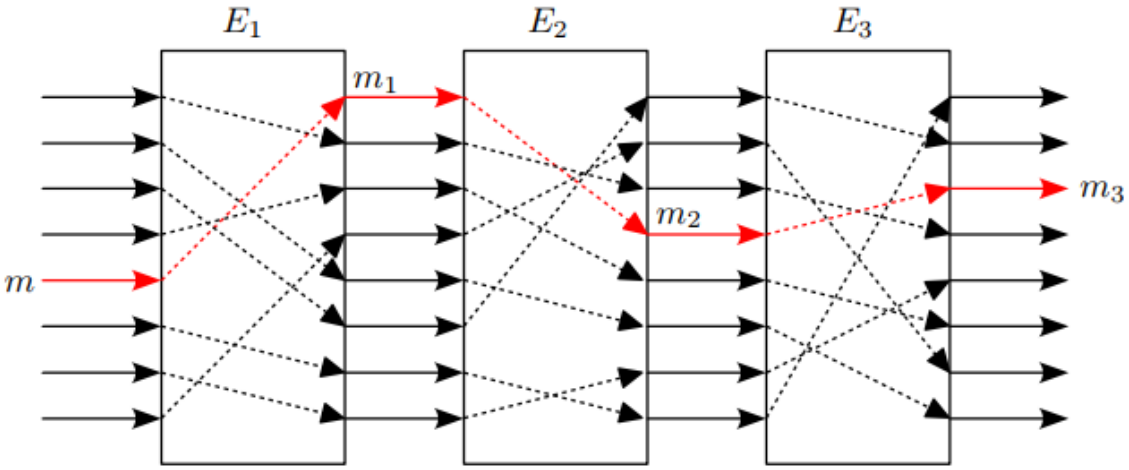

Figura 3.1: Exemplo de uma mix network

Uma das grandes vantagens do uso de *mix networks* é o nível elevado de confiança que estas fornecem. Por exemplo, numa *mix network* com K *mixers* (servidores proxy), se K - 1 forem desonestos a *mix network* continuará, muito provavelmente, a ser segura desde que o *mixer* honesto faça o seu trabalho corretamente [38]. Como referido por Pedro Vasconcelos [38], as *mix networks* não existem por si só, mas sim, existem através de uso de vários outras primitivas como, por exemplo, a encriptação homomórfica e *Zero-Knowledge Proofs* .

3.2 Criptografia Homomórfica

A criptografia homomórfica é um esquema criptográfico que permite trabalhar com dados encriptados, realizando algumas operações, sem a necessidade de os desencriptar. Desta forma, estes esquemas criptográficos permitem minimizar a exposição de informações (especialmente as mais sensíveis). Crê-se que esta foi proposta pela primeira vez em 1978 por Ronald Rivest, Len Adleman e Dertouzos no artigo intitulado de "On Data Banks and Privacy Homomorphisms" [18].

Este tipo de criptografia baseia-se no conceito de homomorfismo matemático que estabelece o seguinte. Considerando dois grupo (A, .) e (B, *) e uma função $F : A \rightarrow B$ que recebe elementos do conjunto A e retorna elementos do conjunto B. Então, F é um homomorfismo se e só se $F(x * y) = F(x) * F(y)$ para qualquer par "x" e "y" pertencentes ao conjunto A. Considerando o conceito referido e mapeando o mesmo para criptografia podemos, por exemplo, considerar o seguinte:

- **A** $\subset \mathbb{Z}$
- '**.**' como as operações que podem ser realizadas nos dados simples
- '*****' como as operações que podem ser realizadas nos dados encriptados
- **F** como a função de encriptação

Considerando que nos dados simples é possível a operação de multiplicação e um algoritmo criptográfico homomórfico capaz de executar essa mesma operação sobre dados encriptados, então podemos facilmente perceber que $F(4*5) = F(4)*F(5)$, ou seja, encriptando o valor

4 e 5 e multiplicando-os o resultado, desencriptado, irá ser igual a 20. Este exemplo pode ser visto na figura 3.2.

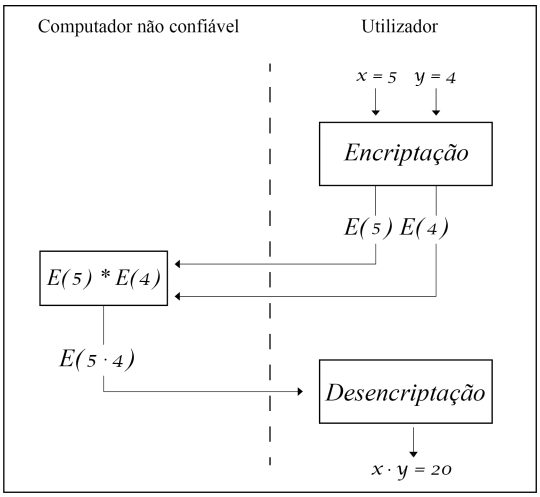

Figura 3.2: Exemplo do funcionamento da criptografia homomórfica

Nota: Uma breve comparação entre criptografia não homomórfica e homomórfica pode ser vista na figura 3.3.

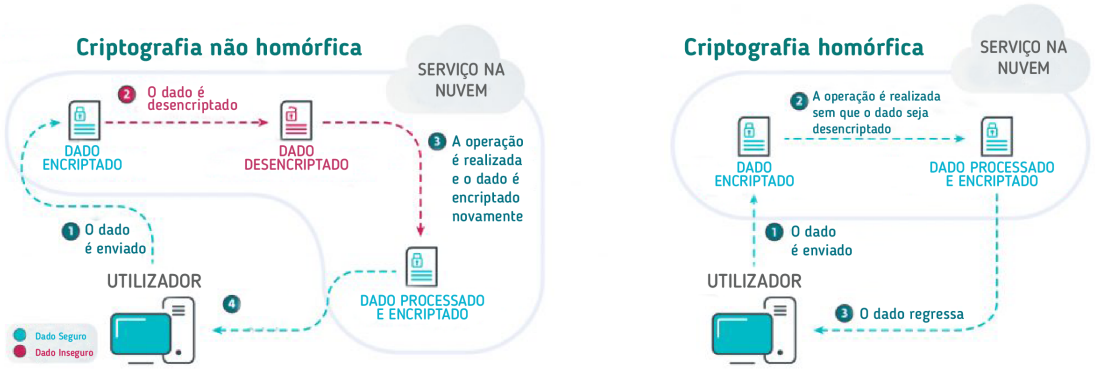

Figura 3.3: Comparação entre criptografia não homomórfica e homomórfica [29]

A proposta dos autores em 1978 sofria, no entanto, de um forte problema. Não ficou demonstrada a viabilidade teórica para criação de um esquema totalmente homomórfico, isto é, um esquema que permite a realização de qualquer tipo de operação sobre os dados encriptados. Somente em 2009, 31 anos depois, é que as bases teóricas para a construção de um esquema totalmente homomórfico [19] foram apresentadas. Craig Gentry foi o responsável por tal proposta aquando a apresentação da sua tese de doutoramento [18]. Desde então, graças ao trabalho da comunidade científica têm-se verificado importantes avanços.

Esquemas criptográficos homomórficos têm várias utilizações, práticas e úteis, das quais se destacam as seguintes: sistemas de votação eletrónica, leilões eletrónicos, computação multipartidária segura e também na área da saúde. Esta última utilização, em particular, é muito interessante e importante pois, por razões legais e de privacidade dos utentes, não se podem divulgar determinadas informações. Por outro lado, essas informações podem ser cruciais para, por exemplo, se estudar uma determinada condição médica. Nestas situ-

ações, os esquemas criptográficos homomórficos podem ser extremamente importantes, em especial os totalmente homomórficos, pois sem revelar os dados permitem o processamento da informação.

3.2.1 Tipos de encriptação homomórfica

Existem essencialmente três tipos de encriptação homomórfica. A principal diferença entre eles está relacionada com o tipo e frequência de operações matemáticas que podem ser feitas sobre texto encriptado. Os tipos de encriptação homomórfica são:

- **parcialmente homomórfica** (*Partially Homomorphic Encryption*) - permite a realização de determinadas operações sobre os dados encriptados. Permite também a realização de uma operação, adição ou multiplicação, sobre os dados cifrados, um número ilimitado de vezes. O *ElGamal*, *Goldwasser–Micali*, *Benaloh*, *Paillier* e o *RSA* são exemplos de sistemas criptográficos parcialmente homomórficos [33].
- **com algum homomorfismo** (*Somewhat Homomorphic Encryption*) - permite a realização de uma operação de adição ou multiplicação, até uma determinada complexidade, um número definido de vezes.
- **totalmente homomórfica** (*Fully Homomorphic Encryption*)- este tipo de encriptação ainda está a dar os primeiros passos. Todavia, têm demonstrado um enorme potencial em várias áreas permitindo compatibilizar privacidade e, ao mesmo tempo, funcionalidade. Tal como os anteriores tipos de encriptação homomórfica, este permite as operações de adição e multiplicação. Este tipo de encriptação permite ainda a realização de operações arbitrárias sobre o texto cifrado. Um dos grandes objetivos da encriptação totalmente homomórfica, é a realização de operações úteis sobre dados encriptados sem a necessidade de partilha de chaves ou segredos. Desta forma, o risco de perda de privacidade e confidencialidade é minimizado.

3.2.2 ElGamal

O ElGamal é um algoritmo criptográfico assimétrico, **parcialmente homomórfico**, proposto por Taher Elgamal em 1984 [16]. Neste, a operação de multiplicação é homomórfica. Baseado na troca de chaves de Diffie-Hellman a segurança deste algoritmo assenta na dificuldade de resolver o problema do logaritmo discreto com elementos de um grupo cíclico finito G [10]. Para explicar o funcionamento do algoritmo iremos recorrer ao Bob e à Alice, onde Bob pretende enviar uma mensagem encriptada para Alice. De seguida apresenta-se o funcionamento geral do algoritmo criptográfico:

Geração de chaves - Para que Bob possa enviar uma mensagem para Alice, este deverá conhecer a chave pública da Alice. Desta forma, Alice deverá gerar o seu par de chaves, privada e pública e partilhar a sua chave pública com Bob. A geração do par de chaves é feita da seguinte forma [32]:

1. Alice deve gerar um número primo p, um número primo grande, e um gerador g do grupo multiplicativo $\mathbb{Z}_p^*$ dos inteiros mod p;
2. Alice deve agora selecionar um número inteiro b aleatório do grupo $\mathbb{Z}$ com a seguinte restrição: $1 \leq b \leq p-2$. Esta será a sua chave privada.
3. Alice pode agora calcular h determinando $g^b \bmod p$.

4. A chave pública de Alice, constituída pelo trio (p, g, h), pode então ser distribuída com o Bob para que este possa encriptar a mensagem que pretende enviar à Alice. Note-se que a chave pública pode ser distribuída publicamente apesar de, para melhor compreensão e no seguimento do exemplo, o Bob tenha sido referido diretamente.

Encriptação - Estando Bob na posse da chave pública de Alice, pode dar início ao processo de encriptação que funciona da seguinte forma [32]:

1. Para que Bob possa enviar uma mensagem encriptada para Alice é necessário representar a mensagem M como um conjunto de inteiros $(m1, m2, ...)$ no intervalo de $\{1, ..., p-1\}$. Estes inteiros m_i serão posteriormente codificados um a um.
2. Bob pode agora encriptar a mensagem M produzindo uma mensagem encriptada C. Para isso, Bob tem de percorrer o conjunto definido no 1º passo e determina c_i, para cada um dos valores o m_i, da seguinte forma:
   a) Selecionar um expoente aleatório k tal que $1 \leq k \leq p-2$. Este expoente deve ser o mais aleatório possível uma vez que descobrir o k dá um conjunto de informação sensível, necessária para desencriptar a mensagem, considerável a um atacante.
   b) Determinar $\gamma = g^k \bmod p$.
   c) Determinar $\delta = m_i * (g^b)^k \bmod p$.
   d) Formar o par $c_i = (\gamma, \delta)$.

Desencriptação - Ao receber a mensagem encriptada C de Bob, Alice precisa agora de a desencriptar para poder ver o seu conteúdo original. Tendo na sua posse a sua chave privada b Alice deverá, para cada c_i, proceder da seguinte forma:

a) Através da chave privada b Alice deverá determinar calcular γ^{p-1-b}. Note que $\gamma^{p-1-b} = \gamma^{-b} = b^{-bk}$.

b) Obter a mensagem m_i calculando $\gamma^{-b} * \delta \pmod p$.

No final, a combinação de todos as m_i, formam a mensagem desencriptada enviada por Alice.

Exemplo:

Bob pretende enviar uma mensagem a Alice com o número '10'.

Alice gera o seu par de chaves escolhendo um número primo aleatório $p = 11$. Desta forma, ela conseguirá representar no máximo o número decimal. Note-se que o número primo escolhido serve apenas para efeitos de demonstração. Numa encriptação segura o número primo deverá ter vários milhares de bits, por exemplo 1024 ou 2048 bits.

Escolhe também o gerador $g = 5$ e $b = 8$ (chave privada). Calcula $h = g^b \bmod p = 5^8 \bmod 11 = 4$

Assim sendo a **chave pública** de Alice é **(11,5,4)**.

Na posse da chave pública de Alice, Bob pode dar início ao processo de encriptação da mensagem. Começa por representar a mensagem M como uma conjunto de inteiros no intervalo {1,..., p - 1} ficando com $m_1 = 10$.

Neste exemplo temos apenas uma m_i pelo que existe apenas uma iteração. De seguida apresentam-se os passos dessa iteração.

1ª iteração:

$$k = 6 \text{ (obtido aleatoriamente)}$$
$$\gamma = g^k \bmod p = 5^6 \bmod 11 = 5.$$
$$\delta = m_i * (g^b)^k \bmod p = 10 * (5^8)^6 \bmod 11 = 7.$$
$$c_1 = (5, 7)$$

Terminadas as iterações a mensagem encriptada é o conjunto de todos os c_i. No caso em concreto, a mensagem é apenas c_1.

Desencriptação

Alice recebeu a mensagem encriptada (5, 7). Para obter a mensagem desencriptada Alice terá apenas que calcular:

$\gamma^{-b} * \delta \pmod p = \gamma^{p-1-b} * \delta \pmod p = 5^2 * 7 \pmod p = 10$

Concluí-se portanto o processo verificando que a mensagem desencriptada é igual à mensagem original enviada por Bob.

3.3 Blind Signatures

Blind Signatures são um tipo especial de assinaturas digitais nas quais a mensagem é ocultada antes de ser assinada [34]. Desta forma, o assinante não consegue ver o conteúdo da mensagem que está a assinar. Assim sendo, se o assinante for posteriormente confrontado com a mensagem não oculta, não será capaz de a relacionar com a mensagem que assinou anteriormente [17]. David Chaum, em 1982, apresentou e propôs as *Blind Signatures* como um componente básico para dinheiro eletrónico [11]. Uma particularidade interessante é que a proposta de David Chaum pretendia aumentar a **privacidade** e **anonimato** nos pagamentos digitais que, nos tempos atuais, faz parte do nosso dia-a-dia.

Este tipo de assinaturas apresenta várias utilizações. Vários protocolos que operam com dinheiro eletrónico têm utilizado este tipo de assinaturas. Todavia, as *Blind Signatures* não se limitam apenas ao dinheiro eletrónico. De facto, estas têm vindo a ser utilizadas em protocolos de votação eletrónica.

***Blind Signatures* na votação eletrónica**

As *Blind Signatures*, como referido, têm vindo a ser aplicadas em alguns protocolos de votação eletrónica. Neste contexto, e sem entrar em grandes detalhes, é geralmente necessário garantir a autenticidade do votante e o anonimato do voto (consideremos apenas estes requisitos). Se, por um lado, o eleitor deve ser autenticado por uma autoridade, também é verdade que essa autoridade não deve ter conhecimento do voto desse eleitor. Neste cenário, as *Blind Signatures* podem apresentar-se como uma possível solução.

Como podemos perceber existem duas entidades presentes: o emissor (o eleitor), que pretende obter uma assinatura do servidor eleitoral, e o "*Prover*" (autoridade eleitoral). O objetivo do eleitor é obter a assinatura da autoridade eleitoral sobre a mensagem que o eleitor lhe passa, sem que a autoridade eleitoral conheça essa mensagem.

Uma visão geral do funcionamento das *Blind Signatures* é apresentado de seguida.

1. O eleitor pretende que a sua mensagem m seja assinada pelas autoridades eleitorais.
2. O eleitor oculta a sua mensagem, geralmente através de um número aleatório denominado de *binding factor*, criando assim a mensagem m'.
3. A mensagem m' é envidada para as autoridades eleitorais juntamente com as credenciais do eleitor.
4. Ao receberem a mensagem m' as autoridades validam as credenciais do eleitor. Caso estejam corretas e reúna condições para exercer o voto, a sua mensagem m' é assinada e enviada ao eleitor. Caso contrário, o processo termina sem qualquer assinatura pelas autoridades eleitorais.
5. Tendo recebido a mensagem assinada, o eleitor verifica se a assinatura é de facto das autoridades eleitorais. Se não for o processo termina. Caso contrário, estando na posse do número aleatório utilizado no 2º passo, reverte a "ocultação" verificando se esta é igual à mensagem m. Caso não seja igual o processo termina pois, as autoridades eleitorais, assinaram uma mensagem que não aquela enviada.
6. Por fim, o seu voto pode ser enviado, juntamente com a assinatura para o servidor eleitoral para ser registado. O servidor eleitoral irá verificar a assinatura e o voto. Se tudo estiver em conformidade, o voto será armazenado para posterior contabilização.

Uma ilustração do processo referido pode ser visto na figura 3.4.

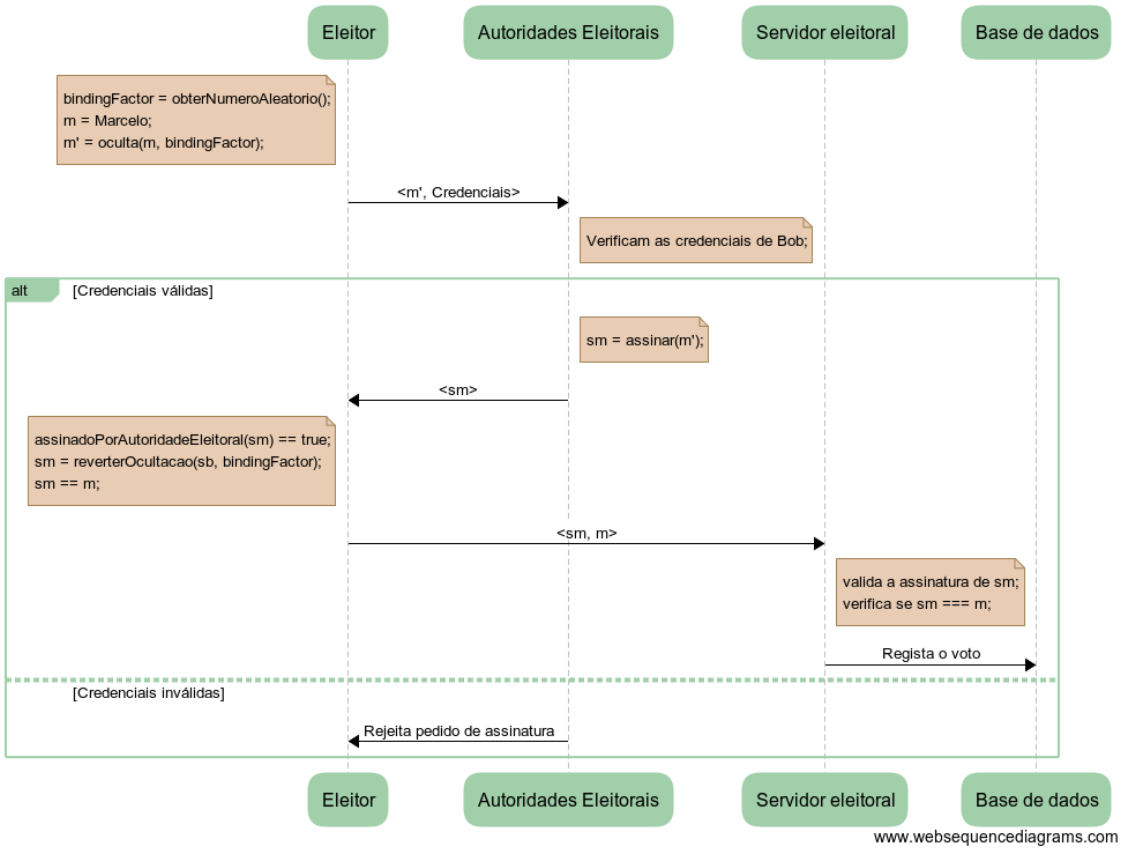

Figura 3.4: Diagrama sequência funcionamento geral das *Blind Signatures*

3.4 Zero-Knowledge Proofs

O origem das *Zero-Knowledge Proofs* remete-nos aos anos 80. Foi nessa altura que os investigadores Shafi Goldwasser, Silvio Micali e Charles Rackoff [20] do reconhecido Instituto de Tecnologia de Massachusetts, mais conhecido por MIT, deram os primeiros passos na criação de um método, nessa altura um método iterativo, que permitia a uma entidade, denominada de "*Prover*", provar perante a outra entidade, denominada "*Verifier*", que uma determinada afirmação é verdadeira sem revelar qualquer outra informação. Note-se que estas provas são sempre probabilísticas. Quer isto dizer que podemos minimizar a probabilidade de o "*Prover*" conseguir fornecer uma prova falsa mas nunca conseguimos ter uma certeza absoluta.

Dentro das *Zero-Knowledge Proofs* podemos encontrar essencialmente dois métodos: os iterativos e os não iterativos. Independentemente do método, uma verdadeira *Zero-Knowledge Proof* precisa de garantir as três propriedades seguintes [38]:

- ***Completeness*** - se uma afirmação for verdadeira, um *Verifier* honesto irá ficar convencido dessa afirmação.
- ***Soundness*** - se uma afirmação for falsa, um *Prover* desonesto não conseguirá convencer o *Verifier* de que a informação é verdadeira.
- ***Zero-Knowledge*** - nenhuma informação é revelada ao *Verifier* a não ser a própria prova.

3.4.1 Métodos Interativos

Os métodos iterativos, tal como o próprio nome indica, requerem interação entre o *Prover* e o *Verifier*. Quantas mais interações existirem, menor é a probabilidade de o *Prover* conseguir enganar o *Verifier* com uma afirmação falsa. O tipo de interações entre o *Prover* e o *Verifier* depende da afirmação que se quer provar. Um simples e informal exemplo é apresentado de seguida novamente com Bob e Alice como personagens.

Contextualização:

Numa caverna circular, com apenas uma entrada, existem dois caminhos, o A e o B, e uma porta a bloquear a passagem entre eles. Esta porta está protegida por um segredo e somente quem descobrir esse segredo consegue fazer a passagem. Alice afirma que sabe o segredo e Bob está disposto a pagar por esse segredo. No entanto Bob apenas fará o pagamento quando estiver certo que Alice sabe de facto o segredo. Por outro lado Alice não revela o segredo até que Bob faça o pagamento. Como se pode facilmente perceber estamos perante um impasse. Como é que este pode ser resolvido?

Para resolver o impasse, Bob e Alice marcam os caminhos da caverna como na figura 3.5. Bob espera que Alice entre na caverna e escolha um dos caminhos.

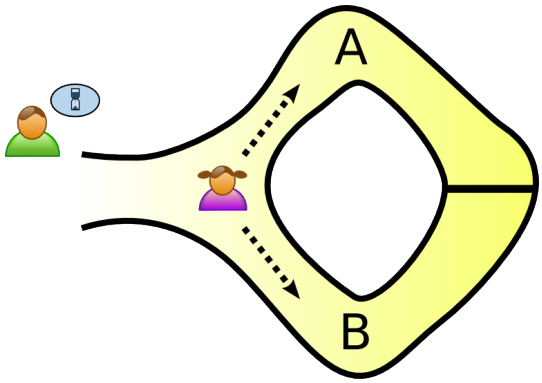

Figura 3.5: Entrada caverna do Ali Baba [31]

De seguida, Bob entra na caverna e grita para Alice sair pelo caminho A ou B. A figura 3.6 exemplifica a entrada de Alice pelo caminho B.

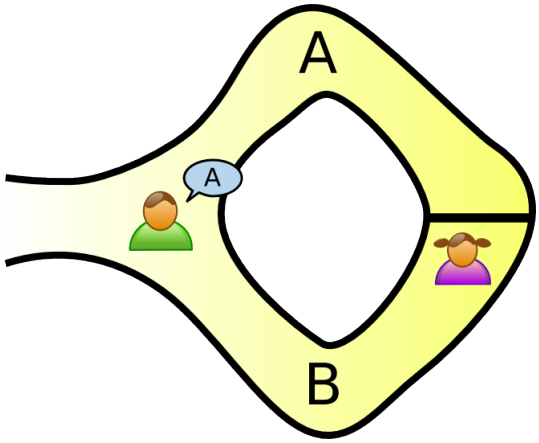

Figura 3.6: Entrada de Alice pelo caminho B na caverna do Ali Baba [31]

Se Alice souber de facto o segredo ela conseguirá sair pelo caminho que Bob lhe indicar.

Como Bob não sabe por qual dos caminhos entrou Alice na caverna, Bob deve repetir o procedimento até que esteja satisfeito com as provas dadas por Alice. Na primeira vez, Alice tem uma probabilidade de 50% de acertar na escolha feita e proferida por Bob. No entanto, se o processo for repetido várias vezes a probabilidade de Alice acertar em todas elas é diminuída de forma considerável e, portanto, se Alice continuar a sair pelo caminho que Bob lhe indica este poderá deduzir que Alice sabe de facto o segredo.

3.4.2 Métodos não interativos

Os métodos não interativos são uma variante dos métodos interativos que permitem fazer prova sem a necessidade de um processo interativo como o verificado anteriormente. De facto, nos protocolos não interativos os *Verifiers* são simuladores que permitem verificar, sem qualquer interação de um utilizador, uma determinada afirmação recorrendo para isso a primitivas criptográficas como *Hashing*.

3.4.3 Casos de uso

As *Zero-Knowledge Proofs* têm um conjunto de aplicações no mundo real bastante grande. Alguns exemplos são:

Em votações eletrónicas - Vários protocolos criptográficos fazem uso das *Zero-Knowledge Proofs* para, por exemplo, provar que os dados do boletim estão preenchidos e representados numa estrutura correta. Helios Voting, Wombat Voting System e VeryVote são apenas três exemplos de sistemas que utilizam as *Zero-Knowledge Proofs*.

Em criptomoedas - ZCash é uma criptomoeda, assente na *blockchain*, que está a tirar partido das *Zero-Knowledge Proofs* para dar maior anonimato, privacidade e segurança aos seus utilizadores. Nesta criptomoeda é utilizada uma versão modificada das *Zero-Knowledge Proofs*, denominada de zk-SNARKS (*Zero-Knowledge Succinct Non-Interactive Argument of Knowledge*).

No setor bancário - O setor bancário começa a tirar partido das tecnologias e adapta-se para fornecer soluções cada vez melhores e competir com a concorrência. Um exemplo é o do banco Holandês ING que implementou uma solução em *blockchain* para permitir aos clientes provar que um determinado número está dentro de um intervalo. Este tipo de solução permite, por exemplo, que um cliente prove ao banco que o seu salário está contido num determinado intervalo sem ter revelar o seu valor exato. A solução criada e implementada, denominada de *Zero-Knowledge Range Proof*, segundo a ING é cerca de 10 vezes mais eficiente que outras *Zero-Knowledge Proofs* o que lhes permite ter um custo operacional menor e, com a vantagem, de trazer mais segurança e privacidade para os clientes.

Como se pode constatar existem várias aplicações distintas. De facto, as enunciadas representam uma quantidade ínfima de possibilidades. O leque destas possibilidades de aplicação é de facto bastante considerável.

3.5 Conclusão

Ao longo do presente capítulo pudemos constatar o papel fundamental que a criptografia desempenha na construção de sistemas de votação electrónica que procuram garantir o cumprimento dos requisitos fundamentais destes sistemas. Nesta secção pretende-se fazer uma breve súmula do mesmo.

Primeiramente abordámos as *mix networks*, que se apresentam como uma boa solução para problemas relacionados com o anonimato. De seguida, abordámos o esquema criptográfico homomórfico no qual é possível realizar determinadas operações sobre dados encriptados fazendo uma breve explicação do seu funcionamento geral , abordámos o algoritmo ElGamal e referiu-se a área médica como uma das áreas que pode beneficiar deste tipo de esquema criptográfico.

Continuando a análise das primitivas criptográficas utilizadas em sistemas de votação, apresentaram-se ainda as *blind signatures*, um tipo especial de assinaturas digitais, em que o assinante não consegue ver o conteúdo que assina pois este está ocultado. Constatou-se que estas têm vindo a ser utilizadas em votação eletrónica e dinheiro eletrónico para aumento da privacidade e anonimato. Por fim, analisaram-se as *Zero-Knowledge Proofs*, um método que

permite a uma entidade provar perante outra que uma determinada afirmação é verdadeira, com uma determinada probabilidade, sem lhe revelar qualquer outra informação. Neste contexto, definiram-se as três propriedades que estas devem garantir e apresentaram-se as suas duas variantes, os métodos iterativos e os não-iterativos, e os casos de uso desta primitiva criptográfica.

Capítulo 4

Análise de Sistemas de Votação Eletrónica

4.1 Scantegrity II

O Scantegrity II é um sistema de votação eletrónica presencial *open source* que foi proposto por David Chaum, Ronald Rivest, Richard Carback, Alan Sherman, Aleksander Essex, Jeremy Clark, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan e Emily Shen. Os autores apresentam, no artigo publicado [12], este sistema como melhoria prática para sistemas de votação de leitura ótica através do uso de códigos de confirmação, impressos com tinta invisível, que constam nos boletins de voto. Este sistema permite manter a privacidade e anonimato do eleitor ao mesmo tempo que permite, a esse mesmo eleitor, poder verificar se o seu voto foi interpretado e contabilizado corretamente.

Em 2009 a cidade de Takoma Park, nos Estados Unidos da América, tornava-se na primeira a utilizar um sistema de votação E2E-V em eleições públicas [3]. No artigo [3], dedicado à implementação desta proposta para a cidade de Takoma Park, os autores apresentam o trabalho desenvolvido, as linguagens, os sistemas óticos utilizados, configurações, processo pré-eleitoral e pós-eleitoral e, por fim, os resultados de um questionário e conclusões.

De acordo com os autores o sistema proposto é constituído pelo seguinte:

Boletim de voto

O boletim de voto proposto é constituído por duas partes. Na primeira parte constam dados da eleição, nos quais se inclui a lista dos candidatos, cada um com um campo de marcação, denominado pelos autores de *bubble*, que será lido através de um leitor ótico. Cada campo de marcação possui uma sequência de caracteres alfanuméricos aleatórios, denominados de códigos de confirmação, impressos com tinta especial, invisível para o olho humano, como se pode ver na figura 4.1 (ao centro). Antes do boletim ser marcado, nenhum do código está visível. Fazendo uso de uma caneta especial, denominada de *decoder pen*, o eleitor marca o(s) candidato(s). Ao fazer isso o(s) código(s) de confirmação são revelados, como se pode ver na figura 4.1 (à direita). À esquerda da figura 4.1 pode-se ver o boletim imprimível com as regiões invisíveis especificadas por um mapeamento de cores falsas (o magenta e o amarelo). Note-se também que se o eleitor o pretender, poderá ficar com um recibo do voto. Neste recibo, que deve ser destacável e conter obrigatoriamente o identificador do boletim, o eleitor deve colocar o código, ou códigos, de confirmação que lhe foram revelados.

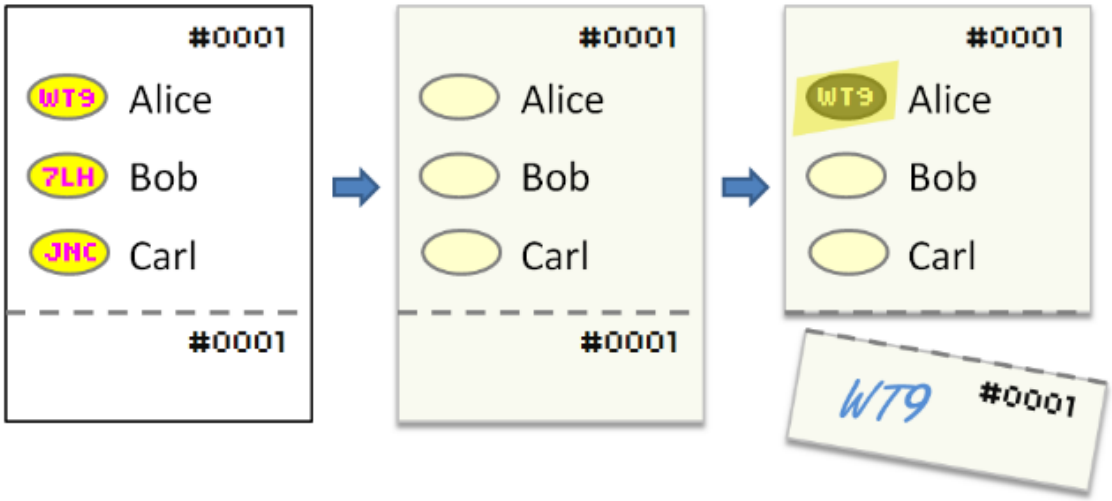

Figura 4.1: Boletim de voto de exemplo [12]

Códigos de confirmação

Como referido anteriormente, dentro das *bubble's* encontram-se códigos de confirmação impressos em tinta invisível. A geração destes códigos requer o cumprimento do seguinte:

- Os códigos de confirmação têm de ser únicos dentro de cada eleição para cada boletim de votação.
- O código de confirmação para cada candidato, dentro de uma eleição e para cada boletim de voto deve ser uniformemente pseudo-aleatório e independentemente selecionado de um conjunto de códigos possíveis (Que pode ser restrito para eliminar determinados caracteres).
- O código de confirmação de um candidato num boletim de voto específico, deve ser secreto e desconhecido ao eleitor até que este marque o candidato no qual deseja depositar o seu voto.

Preparação da eleição

Antes do início do período de votação é necessário realizar-se um conjunto de tarefas. Segundo os autores, os funcionários eleitorais devem fornecer a *seed* [1] ao gerador de números pseudo-aleatórios. No total deverão ser gerados B * N códigos, sendo B o número de boletins de voto e N o número de candidatos, e também as tabelas [12]:

P : Tabela que contém os códigos de confirmação, pela ordem que estes foram gerados. Esta tabela faz a correspondência entre o boletim, o código de confirmação e o candidato. Esta tabela é utilizada na geração da tabela **Q**. Note-se que, em momento algum a tabela Q é publicada.

Q : Tabela na qual os códigos de confirmação em cada linha da tabela **P** foram pseudo-aleatoriamente permutados. Desta forma, não existe uma correspondência fixa entre a coluna e o candidato. Esta tabela será utilizada, após o término da votação, para expor os códigos de confirmação revelados aos eleitores. Esta é pública e deve ser publicada antes do início da votação.

[1] Valor de inicialização do gerador de números pseudo-aleatórios

R : Esta tabela é constituída por três colunas. São elas: Flag, **Q**-Pointer e **S**-Pointer. Na coluna **Flag** será sinalizado, após o término da votação, se foi realizado um voto com o código de confirmação subjacente. As restantes colunas, **Q**-Pointer s **S**-Pointer, funcionam como ponteiros, pseudo-aleatoriamente gerados, para a célula das tabelas Q e S onde está subjacente o mesmo código de confirmação. Note-se que, segundo os autores, esta tabela irá ser utilizada no processo de auditoria através de verificação parcial aleatória (do inglês, *randomized partial checking*).

S : Cada célula da tabela tem subjacente um código de confirmação associado. Inicialmente esta tabela encontra-se vazia. No final do período de votação as células cujos códigos de confirmação subjacentes tenham sido utilizados para efeitos de voto, serão sinalizadas. Nesta tabela, como se pode verificar através da figura 4.2 cada coluna representa um candidato enquanto que cada linha pode representar um qualquer boletim de voto.

Considerando uma eleição com N = 3 candidatos (Alice, Bob e Carl) e B = 5 boletins de voto, as tabelas, antes do período de votação ter iniciado, ficariam como se pode verificar na figura 4.2.

Na referida figura podemos claramente verificar que a tabela **Q** é muito semelhante à tabela **P** com a particularidade de ter ocorrido a permutação e a remoção dos candidatos conforme referido anteriormente. A tabela **R** é aquela que contém os apontadores para as tabelas **Q** e **S**. Para um melhor entendimento do processo, iremos agora recorrer a um exemplo concreto. Analisando a primeira linha da tabela temos o **Q**-Pointer = (Ballot ID, Coluna) = (0005, 1) e o **S**-Pointer = (Linha, Coluna) = (2,1); Quer isto dizer que o código de confirmação subjacente à primeira linha é o que consta na primeira coluna do boletim 0005 da tabela **Q**, ou seja, o **M39** que corresponde a um voto na Alice. Caso este código seja revelado ao eleitor, desde que não seja em um voto de auditoria, um visto estará presente na coluna **Flag** da tabela **R**, e na linha 2 coluna 1 da tabela **S**.

Ballot ID	Alice	Bob	Carl
0001	WT9	7LH	JNC
0002	KMT	TC3	J3K
0003	CH7	3TW	9JH
0004	WJL	KWK	H7T
0005	M39	LTM	HNN

Table **P**

Ballot ID			
0001	7LH	WT9	JNC
0002	J3K	TC3	KMT
0003	9JH	CH7	3TW
0004	KWK	H7T	WJL
0005	M39	HNN	LTM

Table **Q**

Flag	**Q**-Pointer	**S**-Pointer
	(0005, 1)	(2, 1)
	(0003, 3)	(4, 2)
	(0002, 1)	(4, 3)
	(0001, 3)	(3, 3)
	(0001, 2)	(4, 1)
	(0005, 3)	(3, 2)
	(0004, 2)	(5, 3)
	(0003, 1)	(2, 3)
	(0004, 3)	(3, 1)
	(0002, 3)	(1, 1)
	(0001, 1)	(2, 2)
	(0002, 2)	(5, 2)
	(0004, 1)	(1, 2)
	(0003, 2)	(5, 1)
	(0005, 2)	(1, 3)

Table **R**

Alice	Bob	Carl

Table **S**

Figura 4.2: Tabelas **P**, **Q**, **R** e **S** [12]

Processo de votação

De acordo com os autores o processo de votação funciona, de forma resumida, da seguinte forma:

1. Primeiramente o eleitor apresenta-se e efetua a sua identificação perante os funcionários eleitorais. Neste momento os funcionários irão verificar se o eleitor consta no caderno eleitoral. Caso conste, então os funcionários irão dar a possibilidade ao eleitor de receber um ou dois boletins de voto e a caneta especial (*decoder pen*), que lhe permitirá escolher a(s) sua(s) preferência(s) e, consequentemente, revelar o(s) código(s) de confirmação. A receção de um ou dois boletins de voto tem que ver com o facto de ser permitido, ao eleitor, poder realizar um voto de auditoria. Por este motivo, se o eleitor optar por receber dois boletins de voto, este deverá escolher um deles para que seja considerado como voto de auditoria. O eleitor, no boletim de auditoria, deve verificar que todos os códigos de confirmação são únicos sendo para isso necessário que utilize a *decoder pen* para revelar todos os códigos de cada candidato. Este boletim e o seu recibo devem ser marcados, pelos funcionários eleitorais, como "Boletim de auditoria", utilizando por exemplo, um carimbo.

2. O eleitor dirige-se à cabine de voto e procede ao preenchimento dos boletins de voto de acordo com as regras da eleição. Nesta fase usa a *decoder pen* para marcar a sua intenção de voto e revelar, consequentemente, o(s) código(s) de confirmação. De seguida, o eleitor insere o boletim de voto no leitor ótico para que as intenções de voto sejam registadas. Neste processo é verificado se o preenchimento do boletim está de acordo com as regras da eleição e se não existem marcas no boletim, podendo

surgir erros e o boletim de voto ser rejeitado. Caso o boletim esteja corretamente preenchido, as escolhas do eleitor são registadas com o identificador de boletim (*Ballot ID*).

3. Caso o pretenda, o eleitor poderá levar consigo o recibo de voto e posteriormente verifica-lo através de uma página web. Para isso, o eleitor apenas necessita de registar o(s) código(s) de confirmação na parte destacável do boletim de voto e ser marcado, pelos funcionários eleitorais, como "Boletim votado". Este recibo deve ser guardado pelo eleitor pelo menos até no final da eleição.

Processo de verificação do voto

Após o término da eleição, as autoridades eleitorais procedem à publicação das tabelas **Q**, **R** e **S** atualizadas como se pode verificar na figura 4.3. Disponibilizam também uma página web que permite aos eleitores, de forma fácil, rápida, simples e intuitiva, verificar o seu voto. Para isso, o eleitor apenas terá que colocar o *Ballot ID* e verificar que o código de confirmação apresentado corresponde ao código que escreveu no recibo. Se o eleitor quiser auditar todo o processo de votação, também o pode fazer através das tabelas **Q**, **R** e **S**, disponibilizadas publicamente. Na eventualidade de o código de confirmação não corresponder, o eleitor deverá abrir uma disputa, informando qual o código de confirmação do seu recibo, que deverá ser avaliada pela organização eleitoral.

Segundo os autores, as disputas podem surgir essencialmente em quatro circunstâncias principais. São elas: (i) erro do eleitor na transcrição do código de confirmação, (ii) tentativa de colocar em questão a legitimidade da eleição, (iii) um erro de leitura do leitor ótico ou (iv) evidência de comportamento fraudulento. As entidades eleitorais, responsáveis pela análise das disputas que possam surgir, deverão analisar cada caso e perceber em qual das circunstâncias se enquadra, devendo produzir um claro relatório sobre o reportado.

Ballot ID			
0001		WT9	
0002	J3K		
0003		CH7	
0004	KWK	H7T	WJL
0005			LTM

Table **Q**

Flag	**Q**-Pointer	**S**-Pointer
		(2,1)
	(0003,3)	
✓		(4,3)
		(3,3)
✓	(0001,2)	
✓	(0005,3)	
	(0004,2)	(5,3)
		(2,3)
	(0004,3)	(3,1)
	(0002,3)	
	(0001,1)	
	(0002,2)	
	(0004,1)	(1,2)
✓		(5,1)
	(0005,2)	

Table **R**

Alice	Bob	Carl
	✓	
✓		✓
✓		

Table **S**

Figura 4.3: Tabelas **P**, **Q**, **R** e **S** após a eleição [12]

A solução apresentada pelos autores vem contribuir de forma positiva para a integridade dos processos eleitorais. De seguida apresentam-se algumas das suas principais contribuições para os processos de eleição eletrónica presencial.

- Fortes garantias da preservação do anonimato do voto.
- Materialização física do voto. Permite verificação manual se necessário.
- Eleitor pode verificar/auditar o seu voto (Verificabilidade de eleitor).
- Qualquer pessoa ou entidade poderá auditar a eleição tornando-a assim mais transparente e confiável (Verificabilidade Universal).
- Rápida contabilização dos votos.
- Maiores garantias da integridade do processo.
- Imune a coerção e também a ataques de randomização.

4.2 Helios Voting

O Helios Voting é um sistema de votação eletrónica remota *open source* proposto por Ben Adida [6] em 2008. Desde então, o sistema tem vindo a sofrer algumas correções e melhorias estando, atualmente, em Janeiro de 2020, na versão 4. No seu artigo, Ben Adida apresentou o Helios como o primeiro sistema de votação eletrónica através da internet que a permitir o que este designou por *open-audit* (referido por outros autores como E2E-V). Para além da apresentação da proposta teórica, Ben Adida apresenta e disponibiliza ao público a implementação dessa mesma proposta de forma totalmente gratuita e aberta. Desta forma, toda aquela pessoa ou entidade que queira analisar a implementação, auditar a mesma, adaptar a solução ou simplesmente utilizá-la nos seus projetos, pode fazê-lo sem qualquer problema.

Como todo o processo de eleição é realizado remotamente, através da internet, utilizando um *browser*, a participação em eleições, independentemente da sua natureza, passa a ser, para o eleitor, mais fácil e rápida. Note-se no entanto que o sistema Helios não é adequado a eleições em que a coerção seja um problema considerável, uma vez que a proposta do sistema não pretende endereçar este problema. De facto, este sistema de votação é mais adequado para eleições que não sofram de um elevado risco de coerção como as eleições governamentais.

O sistema foi desenhado com base no modelo de Benaloh [9] numa investigação da Microsoft e recorre à aplicação de técnicas criptográficas avançadas, de forma a assegurar a integridade dos votos e a privacidade do eleitor tendo em conta o ambiente inseguro em que este opera (internet). O processo eleitoral, do ponto de vista funcional e de acordo com o artigo original [6], funciona de forma resumida da seguinte forma:

Criação da eleição

Um utilizador registado no sistema Helios procede ao registo da eleição indicando um título, uma data e uma hora de início e de fim da votação entre outras informações. Este utilizador será o administrador desta eleição. Neste momento o sistema gera um novo par de chaves criptográficas, uma privada e outra pública, recorrendo ao algoritmo **El-Gamal**. Este par de chaves criptográficas é gerado para o *Trustee* do servidor eleitoral, denominado de *HeliosTrustee*, que é criado e armazenado num Sistemas de Gestão de Base de Dados (SGBD). Como em todos os sistemas criptográficos assimétricos, a segurança só é garantida se a chave privada for mantida em segurança.

Trustees

Desde a 2ª versão, o HeliosVoting garante a confidencialidade dos votos utilizando encriptação distribuída [35]. Os votos de cada eleitor são encriptados do lado do cliente e só então enviados para o servidor que nunca, existindo pelo menos um *Trustee* para além do *HeliosTrustee*, terá em sua posse a chave de desencriptação uma vez que as chaves criptográficas de cada *Trustee* são geradas por estes nos seus computadores e a sua chave privada nunca é enviada ao servidor eleitoral [35]. Desta forma, para ser possível, de forma ilícita, proceder à desencriptação de votos, é necessário que todos os *Trustees* da eleição partilhem e combinem as suas chaves privadas.

Mas afinal o que são os *Trustees*? Os *Trustees* são entidades confiáveis que integram os processos eleitorais do Helios com o objetivo de manter um elevado nível de confidencialidade dos votos. Estas entidades participam ativamente no processo de desencriptação do resultado da contabilização dos votos. Um qualquer número de *Trustees* pode ser adicionado a uma eleição de forma a aumentar as garantias relativas à preservação da confidencialidade dos votos.

Os *Trustees* desempenham um papel de enorme importância e bastante sensível. Cada um deles é responsável por gerar um par de chaves criptográficas, recorrendo ao algoritmo ElGamal, que serão utilizadas no processo eleitoral para garantir a confidencialidade dos dados e permitir, após a contabilização dos votos encriptados da eleição, desencriptar parcialmente o resultado da contabilização de votos que se encontra encriptado.

Cada *Trustee* tem que enviar para o servidor eleitoral a sua chave pública enquanto que a sua chave privada deve ser mantida em segurança. Todavia, apenas a submissão da chave pública não é suficiente porque o servidor eleitoral precisa de confirmar que a chave privada relacionada é conhecida pelo *Trustee*. A solução encontrada para resolver esta questão, sem ter que se facultar a respetiva chave privada, foi recorrer às *Zero-Knowledge Proofs*. Através destas provas é possível fornecer ao servidor eleitoral não só a chave pública do *Trustee*, assim como fornecer uma prova criptográfica que garante que, no momento da submissão, este conhecia a chave privada relacionada.

Configuração do boletim de voto

Uma das partes inerentes a uma eleição é a configuração do boletim de voto. O utilizador deve indicar as questões que o boletim eletrónico conterá e as suas possíveis respostas. Adotando esta forma de funcionamento, o Helios permite abranger vários tipos de eleições independentemente de terem uma ou mais questões. Uma característica interessante, é a possibilidade de se tornar a disposição das respostas aleatória em cada boletim. Este comportamento permite prevenir desigualdades causadas pelo posicionamento, por exemplo, no topo do boletim.

Registo de Eleitores

O registo de um eleitor é feito de forma muito simples através da indicação do **Nome** e do **Email** que se pretende adicionar no caderno eleitoral. Por cada utilizador registado será automaticamente gerada uma palavra passe com 10 caracteres utilizando um gerador pseudo-aleatório. Todavia, o registo de eleitores pode ser feito recorrendo a outras abordagens. Um exemplo será a utilização de serviços de autenticação externa, como por exemplo os fornecidos pela Google e Facebook ou mesmo através de um Single sign-on (SSO) ou serviço Lightweight Directory Access Protocol (LDAP) de uma entidade [35].

Congelamento da eleição

Uma particularidade interessante no Helios é que o administrador tem que congelar a eleição antes que a mesma possa iniciar. Mesmo que a eleição, segundo as datas, esteja a decorrer, é necessário realizar este processo. Entenda-se por congelar a eleição o seguinte: ato de tornar imutável os dados da eleição, o caderno eleitoral (lista de eleitores), datas de votação e boletim de voto. Neste processo é gerada a chave pública da eleição. A chave pública da eleição é gerada através da combinação de todas as chaves públicas dos *Trustees*. Como tal, o congelamento da eleição só pode ser feito após todos os *Trustees* da eleição terem submetido as suas chaves públicas e comprovado que tinham conhecimento da chave privada relacionada.

Por fim, neste processo é gerado um *hash* criptográfico da eleição (impressão digital da eleição). Desta forma, se os dados da eleição forem alterados, esta situação pode ser facilmente detetada. Este fica disponível publicamente e pode inclusive ser partilhado por vários meios de comunicação. Qualquer pessoa ou entidade que queira verificar se a eleição foi alterada pode fazê-lo. Para isso, terão que obter os dados da eleição, disponíveis para o público, e gerar o *hash* da mesma forma e utilizando o mesmo algoritmo que o HeliosVoting utiliza. No fim basta comparar o *hash* fornecido pelo sistema eleitoral com o que se computou e verificar se são iguais. Caso não sejam quer dizer que algum parâmetro da eleição foi alterado.

Envio de email para participação

Uma vez que o caderno eleitoral esteja completo, o administrador da eleição poderá proceder ao envio do email com as instruções de voto. Este email contém uma descrição da eleição, a password gerada para o utilizador, datas de votação, o SHA-1 dos parâmetros da votação gerado e também o endereço pelo qual o utilizador pode exercer o seu direito de voto.

Processo de votação

De acordo com os autores, o processo de votação funciona, resumidamente, da seguinte forma:

1. Utilizando o link da eleição, enviado para o email, o eleitor acede ao Ballot Preparation System (BPS). Na página de votação, criada utilizando o modelo de Single Page Application (SPA), o utilizador pode exercer o seu direito de voto na eleição para o qual foi convidado. No momento em que o eleitor entra na página de votação, é descarregada toda a informação necessária para que o voto possa ser feito mesmo sem acesso à internet, até ao momento em que deseja submeter o seu voto encriptado (encriptado no *browser*). O *hash* da eleição é também mostrado ao utilizador nesta página.

2. O eleitor regista e escolhe as suas preferências e vai avançando no processo. Toda a mudança de contexto que ocorre ao avançar com o processo, é realizada no *browser* recorrendo a Javascript. Como as informações estão em memória, se o eleitor decidir fechar a página, todas as respostas serão perdidas e será necessário iniciar o processo de raiz.

3. Após o eleitor ter completado o preenchimento do formulário, é-lhe solicitado que verifique o preenchimento e, se necessário, proceda às alterações que possam ser necessárias. Se estiver tudo conforme, o eleitor pode "selar" o voto. Neste momento, o voto é encriptado e é gerado o *hash* criptográfico utilizando SHA-1. De seguida o eleitor pode fazer uma de duas coisas:

- **Auditar o voto** - Escolhendo auditar o voto será indicada ao eleitor de que forma o voto foi encriptado. Algumas implementações fornecem também informações de como é que o utilizador pode replicar a encriptação de forma a que possa ter certeza que a encriptação foi realiza corretamente. Se após este processo o eleitor pretender submeter o seu voto, o sistema irá encriptar o voto com outros parâmetros.
- **Submeter o voto** - Escolhendo submeter o voto, o sistema, através de Javascript, remove todos os dados não encriptados do *browser* e solicita ao eleitor que se autentique perante o sistema de votação. Autenticando-se com sucesso o voto encriptado do eleitor é enviado para o sistema de votação. Se tudo estiver correto, o servidor responde com uma mensagem de sucesso e todos os dados (mesmo aqueles encriptados) são removidos do *browser* do eleitor. Neste momento é também enviado um email para o eleitor a confirmar o seu voto juntamente com o *hash* criptográfico gerado. Caso o pretenda, o eleitor pode, diretamente através do *browser*, gerar um recibo de voto onde também consta o *hash* criptográfico.

Criptografia no browser

Como vimos anteriormente o voto é encriptado no terminal do cliente através do browser. Em 2009, na apresentação da proposta, os autores recorriam ao LiveConnect para aceder à Java Virtual Machine (JVM) através do Javascript. A JVM permitia a obtenção de dados aleatórios seguros e também a possibilidade de se executarem cálculos computacionalmente exigentes como os requeridos para a encriptação dos dados. No entanto, esta abordagem acabou por se revelar um problema muito por causa da indisponibilidade da JVM e da falta de suporte em alguns *browsers* [35].

Ao longo dos anos a performance dos interpretadores de Javascript incorporados nos *browsers* aumentou consideravelmente o que permitiu fazer os cálculos necessários diretamente através de Javascript [35]. Desta forma, a JVM deixou de ser um requisito. De facto, na atualidade, o Helios recorre a implementações criptográficas desenvolvidas totalmente em Javascript.

Processo de verificação do voto

A verificação dos votos é uma etapa com uma relevante importância para a integridade da votação. De forma a que o processo de verificação do voto fosse, para os eleitores, o mais fácil possível, Ben Adida desenvolveu um programa para verificar apenas um voto e um outro para verificar toda a eleição. De seguida, apresenta-se o processo proposto e implementado por Ben Adida.

- **Verificação de apenas um voto** - Para a verificação de um voto, o programa desenvolvido recebe como parâmetro a estrutura JSON que é retornada pelo processo de auditoria de voto que pode ser feito pelo eleitor antes de submeter o voto. Nesta estrutura estão dados como: voto encriptado, voto em claro, fator de aleatoriedade e o identificador da eleição. Com estes dados, o programa de verificação mostra ao eleitor o *hash* da eleição, do texto encriptado e os dados em claro do boletim de voto. Desta forma o eleitor pode comparar os dados que este programa lhe mostra com os dados que lhe são mostrados no passo final da votação.
- **Verificação da eleição** - O programa desenvolvido para a verificação de uma eleição recebe o identificador da eleição a verificar. Automaticamente, são obtidos os dados da eleição, os boletins de votos, os votos, as provas criptográficas, etc. Com

estes dados o programa de verificação valida todas as provas criptográficas e votos e recalcula os resultados da eleição.

Ben Adida apresentou uma solução que aumenta o nível de segurança e privacidade dos eleitores. Através de mecanismos e técnicas criptográficas avançadas, a solução permite não só armazenar a informação de forma segura e garantir a privacidade do eleitor, assim como também dá a possibilidade de o eleitor verificar/auditar o seu voto ou até mesmo a eleição por completo. Como os dados da votação ficam publicamente visíveis, existe uma maior transparência que acaba por se refletir na confiança dos eleitores, uma vez que estes estão sujeitos ao escrutínio público.

Ao longo dos vários anos, desde a apresentação da proposta à implementação original, o Helios tem sofrido várias alterações e correções que visam melhorar a sua segurança. Uma das mais claras é a atualização do algoritmo de *hashing* que na versão original era o SHA-1 e que atualmente é considerado inseguro. O envio das credenciais feito através de email é considerado inseguro. A utilização de serviços de autenticação externa, como os referidos anteriormente, poderão mitigar este problema. Outras falhas foram vindo a ser descobertas, e resolvidas, ao longo dos anos através de investigações levadas a cabo por vários investigadores.

4.3 EVIV

O EVIV é um sistema de votação eletrónica remota E2E-V proposto pelos investigadores Rui Joaquim, Paulo Ferreira e Carlos Ribeiro em 2012. Esta proposta nasce do advento da internet e dos estilos de vida moderna, que exigem cada vez mais uma maior mobilidade, com o objetivo de fornecer um sistema de votação pela internet que seja confiável, garanta a integridade do voto e da eleição, ofereça total mobilidade aos eleitores e preserve a privacidade do eleitor mesmo em terminais públicos e/ou inseguros como é o caso dos computadores disponíveis em bibliotecas, cafés e espaços internet [24]. Como referido pelos investigadores o EVIV fornece adicionalmente mecanismos de verificação do voto nos quais o eleitor apenas precisa de corresponder duas pequenas cadeias de caracteres que, segundo os investigadores, são utilizadas na deteção e proteção contra manipulações dos votos.

Ao contrário de várias propostas, nas quais se inclui o Helios, esta não exige que os terminais dos eleitores sejam seguros. De facto, a grande maioria destes terminais não são considerados minimamente seguros pois podem ser facilmente comprometidos através de um conjunto diverso de ataques e/ou técnicas. Assim sendo, esta proposta traz já uma clara vantagem. À semelhança do que se verifica no Helios, esta proposta, para prevenir algumas formas de coerção, permite que um eleitor possa submeter vários votos mas que apenas seja considerado o último.

Os investigadores, na sua proposta, propõem a utilização de um dispositivo, denominado de Voter Security Token (VST), através do qual são gerados os códigos de voto, para cada candidato, e também através do qual o voto do eleitor é encriptado. Dadas as capacidades de computação limitadas dos VST, os investigadores desenvolveram o MarkPledge 3 [25] que é uma especificação menos exigente do algoritmo MarkPledge. A proposta faz ainda uso da técnica de *code voting* que é uma técnica utilizada em várias propostas de votação eletrónica remota como objetivo de resolver o problema da votação através de terminais inseguros [26].

Preparação da eleição

Antes do dia da eleição é necessário proceder ao registo dos eleitores numa base de dados oficial. Em Portugal, o processo de registo de um cidadão na lista de eleitores é, neste momento, automático quando este reúna as condições exigidas por lei. Após este registo à pessoa, agora denominado de eleitor, é-lhe facultado um VST que contém um par de chaves criptográficas [24] que será utilizado para autenticar o eleitor nas eleições.

Aquando a criação da eleição, onde são definidos um conjunto de atributos importantes, devem também ser definidos os *Trustees* da eleição. Este processo e a necessidade dos mesmo foi explicada anteriormente na proposta Helios pelo que aqui apenas se refere a possibilidade e reitera a importância da existência destas entidades.

Antes de cada eleição, os eleitores conectam o seu VST ao seu terminal e acedem ao serviço de registo de registo eleitoral no qual manifestam o interesse em participar numa dada eleição. Após o registo, o VST do eleitor gera um código, para cada candidato, e um código de confirmação de voto que, no conjunto, formam o denominado *code card*. Note-se que os códigos gerados pelo VST são pequenas cadeias com 4 ou 5 caracteres. Na tabela 4.1 pode ver-se um exemplo de um *code card*.

Code Card	
Candidato	**Código de voto**
Leandro	PK8Y7
Marcelo	LKOTT
Confirmation Code **YJ8F**	

Tabela 4.1: Code Card exemplo

Processo de votação

Chegado o dia, ou dias, das eleições, os eleitores podem então exercer o seu direito de voto. Este procedimento requer que o eleitor esteja na posse do seu VST e do seu *Code Card*. O funcionamento deste processo funciona, resumidamente, da seguinte forma:

1. O eleitor conecta o seu VST ao terminal de onde deseja votar (por exemplo ao seu computador), introduz e envia o código do candidato em que deseja votar para o VST.
2. O VST encripta e gera o recibo do voto. Através do terminal do eleitor, estes dados são enviados para o serviço de urna digital designado na proposta por *Ballot box service*.
3. Por fim, o terminal mostra ao eleitor o recibo de voto. Este deverá confirmar se o *Confirmation Code*, que consta no *Code Card* do eleitor, está associado ao candidato seleccionado.

Um exemplo do processo de seleção do candidato e da verificação do recebido de voto pode ser encontrado na figura 4.4.

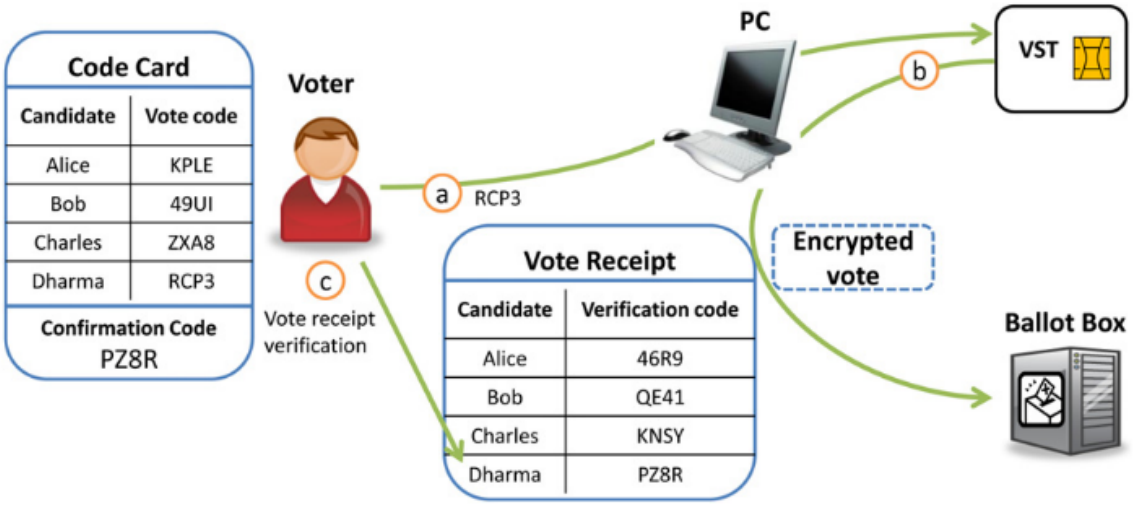

Figura 4.4: Seleção de candidato e verificação do recibo de voto [24]

Término do período de votação

Após o fim do período de votação os votos encriptados e seus recibos são disponibilizados publicamente. Assim sendo, um eleitor poderá facilmente confirmar se o seu voto foi contabilizado analisando o recibo do voto que efetuou.

Contabilização dos votos

Segundo os investigadores, o processo de contabilização dos votos é realizado através de técnicas e algoritmos que garantem a privacidade e a verificabilidade. Neste processo os *Trustees* da eleição, realizam uma contabilização anónima e homomórfica dos votos. À semelhança do que acontece no Helios, as provas criptográficas geradas no processo, são disponibilizadas publicamente para que qualquer pessoa ou entidade possa verificar a corretude dos resultados eleitorais.

Processo de verificação do voto

Após o término do período de votação, todos os dados públicos da eleição podem ser verificados. Entre estes dados estão os boletins de voto, os recibos e o próprio resultado eleitoral. O processo de verificação proposto é realizado em duas etapas. São elas:

- Verificação dos dados da eleição - Através de algoritmos criptográficos as entidades independentes, interessadas em garantir o correto funcionamento do processo eleitoral e dos resultados, irão verificar a validade dos votos e dos seus recibos. Também os próprios eleitores podem verificar os seus votos (inclusive dentro do período de votação). Para isso, devem verificar se no recibo do seu voto o candidato que escolheu tem o *Confirmation Code* que consta no *Code Card* do eleitor.
- Verificação do resultado da eleição - Para garantir que os resultados eleitorais estão corretos, o EVIV permite que as partes interessadas possam recalcular os resultados. Tal é possível graças à encriptação homomórfica que permite, mesmo com os votos encriptados, proceder à contabilização dos mesmos.

Comparação do EVIV com outros sistemas de votação eletrónica remota

Segundo os autores do EVIV, e com os quais se concorda, o sistema proposto apresenta grandes melhorias face a sistemas bem conhecidos e utilizados (um dos quais, VeryVote, dos

mesmos autores). Na figura 4.5 apresenta-se a comparação, feita pelos autores, com outros sistemas End to End (E2E). Nesta pode constar-se que, comparativamente ao Helios, o EVIV oferece uma autenticação do eleitor mais forte, privacidade e segurança mesmo em terminais inseguros e maiores garantias de que o voto foi interpretado e contabilizado corretamente.

	Internet E2E					
	Helios (v3)	PGD	VeryVote	SCV	Heiberg et al.	EVIV
Highly sound voter cast-as-intended verification	×	×	✓	×	✓	✓
Cast-as-intended verification resistant to the collusion of all system components	✓	×	✓	×	×	✓
Protects voter's privacy from a compromised voting PC	×	✓	✓	✓	×	✓
Voters' privacy is not broken by a simple collusion of system components	✓	✓	×	×	×	✓
Universal tally verification	✓	✓	✓	✓	✓	✓
Full voter's mobility	✓	×	×	×	×	✓
Strong voter's authentication	×	×	×	×	✓	✓

Figura 4.5: Comparação do EVIV com outros sistemas de votação eletrónica remota [24]

4.4 Conclusão

Ao longo do presente capítulo analisaram-se três sistemas de votação eletrónica. Neste contexto, apresentou-se o funcionamento geral destes sistemas assim como as vantagens e desvantagens de cada um.

Introduzimos o capítulo abordando um sistema de votação eletrónica presencial, o Scantegrity II, sistema esse utilizado já em eleições nos Estados Unidos da América. De seguida, apresentou-se o sistema de votação eletrónica remota, o Helios, bastante utilizado em diversas eleições por todo o mundo. Este sistema foi, ao longo dos anos, escrutinado por investigadores académicos e profissionais das áreas de segurança, criptografia e matemáticas, onde várias contribuições ajudaram a tornar o Helios ainda mais seguro. Por fim, realizou-se uma breve análise de um sistema de votação eletrónica remota, proposto por três investigadores Portugueses, o EVIV, que pretende elevar o nível de privacidade, segurança e demais requisitos dos sistemas eleitorais a um novo patamar, tendo em consideração os terminais inseguros e a importância da mobilidade nos dias de hoje.

O Scantegrity II, como pudemos constatar, é um sistema de votação eletrónica presencial, ou seja, um sistema que exige a presença física de um eleitor num dado local e num dado período. Este é um sistema muito simples e bastante semelhante ao sistema de votação tradicional em urna, sendo que as principais diferenças residem no facto de a caneta para marcação do candidato, no qual deseja votar, possuir um reagente químico que irá revelar o código de confirmação associado à sua escolha. Outra diferença é o facto de ter um recibo de voto com o código do candidato escolhido. Sendo este um sistema E2E-V o eleitor, após o término da eleição, poderá verificar se o seu voto foi interpretado e contabilizado pelo sistema corretamente. Em relação aos sistemas tradicionais este fornece uma grande vantagem que é a rapidez na obtenção dos resultados dos votos e a possibilidade de o eleitor poder verificar o seu voto após o término da votação.

O Helios Voting, sistema de votação eletrónica remota, é um sistema *open source* que tem vindo a ser utilizado em eleições por todo o mundo. Ao contrário do Scantegrity II este sistema foi especialmente proposto para votações, de forma segura, através da internet. Este é um sistema que faz uso de técnicas e algoritmos criptográficos avançados para garantir a privacidade do eleitor, a integridade do voto e a verificabilidade por qualquer pessoa/entidade. Através dos *Trustees*, este sistema consegue garantir que nem o próprio

servidor tem capacidade, a não ser que exista conluio entre todos os *Trustees* e também entre o *HeliosTrustee* (*Trustee* por omissão das eleições), de desencriptar qualquer um dos votos. Todavia, recorrendo à criptografia homomórfica este sistema consegue produzir o resultado da eleição mesmo sem desencriptar os votos.

O processo de votação no Helios é também ele muito simples e intuitivo para os eleitores e a verificação dos votos, através das plataformas criadas para o efeito, são também muito simples. Em relação aos sistemas tradicionais, este sistema apresenta como vantagens a mobilidade, a rapidez na contabilização dos votos, a acessibilidade e a verificabilidade que oferece. Por outro lado, este sistema não garante a não-coercibilidade, não existe uma evidência física do voto, não tem em conta o problema dos terminais inseguros e, para o eleitor, é um processo mais obscuro.

Por fim, procedeu-se à análise do EVIV, um outro sistema de votação eletrónica remota E2E-V, que pretende resolver alguns problemas verificados geralmente nestes sistemas. Um destes problemas é o dos terminais inseguros dos eleitores. Uma grande vantagem deste sistema em relação ao Helios é precisamente o facto de não "exigir" um terminal seguro para garantir a privacidade e segurança do voto. Tal como no Helios este sistema recorre a várias técnicas criptográficas, como *code voting*, *Trustees* e criptografia homomórfica, para satisfazer os requisitos da votação independentemente do terminal.

Para fornecer toda esta segurança, este sistema exige, em contrapartida, que o eleitor possua um VST que incorpora um par de chaves criptográficas para a encriptação e assinatura do voto. Tal como no Helios, esta proposta não necessita de desencriptar qualquer voto para proceder à contabilização dos resultados pois recorre a algoritmos criptográficos homomórficos. Para determinadas eleições, que apresentem um risco elevado de tentativas de manipulação, esta proposta apresenta-se como a mais adequada, uma vez que apresenta uma autenticação do eleitor mais forte, permite o voto privado e seguro mesmo em terminais inseguros e uma verificação de que o voto foi interpretado e contabilizado corretamente.

Capítulo 5

Orçamentos participativos

Os orçamentos participativos são mecanismos de democracia participativa, que dão aos cidadãos o poder de decidir ou influenciar decisões de como e onde serão aplicadas as verbas dos orçamentos públicos. Atualmente, existem dois grandes tipos de orçamentos participativos: **os orçamentos participativos consultivos**, nos quais os cidadãos são auscultados, mas cuja decisão cabe ao promotor do orçamento; **orçamentos participativos deliberativos**, nos quais os cidadãos apresentam propostas e decidem através do voto, onde serão investidas as verbas destinadas ao orçamento participativo.

Segundo consta no *Participatory Budgeting World Atlas* de 2019, mundialmente, na altura da recolha dos dados, existiam entre 11690 e 11825 orçamentos participativos distribuídos pelo continente Americano, Europeu, Asiático, Africano e Oceania. Estes números mostram claramente que os orçamentos participativos são um mecanismo importante para a sociedade atual. De facto, os decisores políticos têm-se apercebido disso e têm vindo a promover a participação dos cidadãos através deste mecanismo.

5.1 Fases de um orçamento participativo

Um orçamento de tipologia deliberativa, apresenta, normalmente 7 fases.

- **1ª Fase** - Aprovação da verba do orçamento público afeta ao orçamento participativo, definição do calendário e a metodologia.
- **2ª Fase** - Apresentação de propostas por parte dos cidadãos. Geralmente a apresentação destas propostas é feita por meios digitais e/ou através de reuniões públicas abertas a todos os cidadãos.
- **3ª Fase** - Análise técnica das propostas. Nesta fase as equipas responsáveis pelo processo do orçamento participativo verificam se as propostas cumprem os critérios de elegibilidade. Caso não respeitem estes critérios são rejeitadas (com a devida fundamentação). Se, por outro lado, a proposta cumprir os critérios, então esta é transformada em projeto.
- **4ª Fase** - As propostas rejeitadas na análise técnica podem ser alvo de reclamações pelos seus proponentes.
- **5ª Fase** - Nesta fase, os projetos irão ser alvo de votação por parte dos cidadãos. Estes votam nos projetos que querem ver concretizados, de acordo com as regras estabelecidas para o orçamento participativo.

- **6ª Fase** - Finda a fase de votação, são apurados os vencedores e publicados os resultados.
- **7ª Fase** - Após a divulgação de resultados, é feita uma análise de todo o processo, de modo a identificar pontos a melhorar no próximo orçamento participativo.

Nota: Alguns orçamentos participativos podem diferir ao nível das fases. As supra-referidas são as fases mais comuns e geralmente utilizadas. Todavia, a título de exemplo, um orçamento participativo poderá ter votação em propostas. Desta votação sairão as propostas que serão então consideradas na análise técnica.

5.2 Sistemas de votação

Nesta secção pretende-se apresentar os sistemas de votação mais utilizados para o exercício do voto em orçamentos participativos de âmbito municipal em Portugal. Note-se que, num orçamento participativo, pode ser utilizado mais do que um sistema de votação. Por exemplo, podem ser utilizados simultâneamente:

- **Tradicional em urna** - Num local e data definidos, os cidadãos comparecem para exercer o seu direito de voto. O funcionamento deste processo encontra-se descrito na secção 2.3. Os resultados podem depois ser colocados em plataformas informáticas para a sua disponibilização ao público.
- **Eletrónico presencial** - Tal como o sistema tradicional em urna, neste sistema o cidadão deverá comparecer num local e data definidos para exercer o seu direito de voto. Neste sistema de votação, meios eletrónicos estarão disponíveis aos cidadãos para exercer o direito de voto. O funcionamento deste processo encontra-se descrito na secção 2.4. Os meios eletrónicos à disposição dos cidadão poderão comunicar com um servidor, local ou remoto, para o registo do voto e, opcionalmente, imprimir um recibo do voto.
- **Eletrónico remoto** - Atualmente, em orçamentos participativos que dispõem desta forma de votação, o voto eletrónico é o mais utilizado. Tal facto prende-se com a facilidade, rapidez e mobilidade que estes sistemas permitem ao cidadão. A partir de um qualquer lugar, um cidadão elegível pode exercer o seu direito de voto. Uma análise mais detalhada pode ser encontrada na secção 2.5. Na votação eletrónica remota os cidadãos têm geralmente ao seu dispor os seguintes canais de votação: **online** (através de uma plataforma web ou aplicação) e/ou **sms**.

5.3 Caderno eleitoral

No que diz respeito ao registo no caderno eleitoral dos eleitores que reúnem condições para o exercício do voto verificou-se o seguinte:

- **Registo pré-votação** - Nesta modalidade os cidadãos que pretendam participar na votação do orçamento participativo em causa, manifestam o interesse durante o período definido (antes do início do período de votação). Nesta fase, são geralmente solicitados dados e documentos para comprovar a sua identidade. Todavia, mais

documentos podem ser solicitados, por consequência das regras de votação dos orçamentos participativos. Se um orçamento participativo for direcionado a pessoas que residam, trabalhem ou sejam estudantes num determinado concelho, são geralmente pedidos documentos que o comprovem.

- **Registo durante o período de votação** - Nesta modalidade os cidadãos que pretendam participar na votação não precisam de se registar antes do início do período de votação. Por norma, neste tipo de modalidade, os cidadãos registam-se (facultando dados e documentos) e votam no mesmo período. Existem pelo menos duas abordagens que podem ser utilizadas com esta modalidade. A primeira é permitir o registo, durante o período de votação, e também o voto, sendo que neste caso o voto fica "pendente" da validação do cidadão. A segunda abordagem possível é permitir que os cidadãos se registem, até N dias antes do término da votação, mas apenas permitir o voto após validação do cidadão.

5.4 Regras de votação

Os orçamentos participativos têm regras muito próprias, no que diz respeito à participação dos cidadãos elegíveis para votar. Nesta secção pretende-se dar a conhecer algumas dessas regras. As regras aqui referidas surgiram da análise realizada a quarenta e oito orçamentos participativos que decorrem em território nacional (continente e ilhas) e que utilizam a plataforma de orçamentos participativos da entidade acolhedora deste projeto.

- **N votos (em projetos diferentes)** - Votações com esta regra permitem que um cidadão elegível vote em, no máximo, N projetos.
- **N votos (com possibilidade de mais do que um voto no mesmo projeto)** - Votações com esta regra permitem que um cidadão elegível vote em, no máximo, N projetos. Todavia, nesta configuração, o eleitor poderá despender mais do que um voto num mesmo projeto.
- **Um voto por projeto** - Votações com esta regra permitem um voto por cidadão elegível em cada um dos projetos a votação.
- **N votos por grupo** - Votações com esta regra definem grupos, constituídos por projetos, e permitem N votos por cada grupo.
- **N votos por área temática** - Votações com esta regra permitem N votos em projetos de uma determinada área. Considerando a existência de duas áreas temáticas, **A** e **B**, um orçamento participativo pode dar direito, por exemplo, a um voto num projeto da área temática **A** e a dois votos num projeto da área temática **B**.
- **N votos por freguesia** - Votações com esta regra permitem N votos por projetos de uma determinada freguesia. Considerando a existência de duas freguesias, **A** e **B**, um orçamento participativo pode dar direito, por exemplo, a dois votos em projetos da freguesia *A* e a três votos em projetos da freguesia **B**.
- **N votos por patamar** - Votações com esta regra definem patamares para cada projeto e permitem que sejam feitos N votos em projetos de um determinado patamar. Os patamares são geralmente definidos em função do orçamento para execução de um projeto. Considerando um orçamento participativo com dois patamares, dos 0 € aos 25.000 € e o outro patamar para projetos com orçamento superior a 25.000 €,

este pode dar direito, por exemplo, a dois votos em projetos enquadrados no patamar dos 0€ até aos 25.000€ e apenas um voto para projetos superiores a 25.000€.

- **Votos com pontuações** - Votações com esta regra geralmente dão ao cidadão elegível a possibilidade de votar, em mais do que um projeto, atribuindo uma pontuação a cada projeto em que deseja votar. Por exemplo, um orçamento participativo pode permitir três votos sendo que o primeiro voto vale X pontos, o segundo Y pontos e o terceiro Z pontos para $X, Y, Z \subset \mathbb{Z} : X > Y > Z$.
- **X votos positivos e/ou Y negativos** - Alguns orçamentos participativos permitem votos negativos, para que os cidadãos possam votar negativamente num projeto. A título de exemplo, verificou-se que alguns orçamentos participativos permitem que os cidadãos façam dois votos positivos ou um voto negativo.
- **N votos obrigatório(s)** - Votações com esta regra impõem ao cidadão elegível a obrigatoriedade de terem que usar N votos. Nos orçamentos analisados esta regra exigia sempre que os votos fossem feitos em projetos distintos.

Resultados da análise

Para uma análise mais detalhada procedeu-se à contabilização, para cada uma das regras suprareferidas, do número de orçamentos participativos que as utilizam. A tabela 5.1 apresenta os resultados constatados.

Regra	**Nº ocorrências**
N votos (em projetos diferentes)	33
N votos (com possibilidade de mais do que um voto no mesmo projeto)	1
Um voto por projeto	1
N votos por grupo	4
N votos por área temática	1
N votos por freguesia	2
N votos por patamar	1
Votos com pontuações	1
X votos positivos e/ou Y negativos	2
N votos obrigatório	2

Tabela 5.1: Orçamentos participativos por regra

Pela tabela anterior, podemos constatar que a grande maioria dos orçamentos participativos analisados, cerca de 69%, utiliza a regra dos N votos em projetos distintos. Esta análise apresenta uma grande importância para se poder prôpor uma arquitetura que permita estas regras, se não todas, pelo menos as de maior importância e que chegam a um maior universo orçamentos participativos.

5.5 Funcionamento do processo de votação

Nesta secção pretende-se dar a conhecer o funcionamento do processo de votação, de forma geral, nos orçamentos participativos nacionais. Da análise feita verificou-se que existem pelo menos duas formas distintas:

A primeira, mais comum, é o voto projeto a projeto. Neste caso, o voto é submetido individualmente ao servidor eleitoral. O servidor eleitoral, ao receber o voto, irá executar

um conjunto de validações para garantir o cumprimento das regras de votação. Se o voto cumprir com as regras então é armazenado na "urna" eletrónica do servidor eleitoral para posterior contabilização.

A outra forma é, o que se denominou, de votação singular, isto é, votação em que o eleitor tem que escolher todos os projetos em que deseja votar, adicionando os mesmos ao boletim de voto, e só depois submete esse boletim ao servidor eleitoral. Ao receber o voto, o servidor eleitoral irá executar as validações necessárias para garantir o cumprimento das regras de votação. Garantido o cumprimento destas regras, o servidor eleitoral armazena os votos na urna eletrónica para posterior contabilização.

Após o término do período eleitoral são determinados os vencedores, em função das regras dos orçamentos participativos, e são publicados os resultados pelas autoridades eleitorais. Após esta fase, existe ainda o acompanhamento dos projetos vencedores que, para o contexto desta dissertação, não apresenta qualquer relevância.

Nota: Um facto curioso, que foi possível constatar durante a realização desta dissertação, é que cerca de 70% dos eleitores registados nas plataformas da entidade acolhedora, em eleições com mais do que um voto por eleitor, utilizam os vários votos com intervalos inferiores a 5 minutos. Num horizonte temporal de 30 minutos constata-se um aumento para cerca de 84%.

5.6 Conclusão

Ao longo deste capítulo abordaram-se os orçamentos participativos começando-se por apresentar os dois grandes tipos, consultivo e deliberativo, referindo a sua importância por todo o mundo. Neste contexto, constatou-se que existem milhares de orçamentos participativos distribuídos pelos vários continentes. Depois, começou-se por analisar de forma geral as fases, os sistemas de votação, os meios geralmente utilizados e a constituição do caderno eleitoral, que pode ser constituído previamente ao período de votação ou durante esse mesmo período.

Na secção 5.4 procedeu-se à análise das regras de votação que geralmente são utilizadas nos orçamentos participativos em Portugal (Continente e Ilhas). Desta análise, constatou-se a existência de 10 regras distintas, umas com maior prevalência que outras, tendo-se apercebido que a grande maioria dos orçamentos, cerca de 69%, permite N votos em projetos distintos. Já os votos por patamar, N votos (com possibilidade de voto no mesmo projeto), um voto por projeto, N votos por área temática e votos com pontuações são das regras menos utilizadas.

Por fim, procedeu-se a uma breve descrição do funcionamento do processo de votação. Através desta análise, pôde constatar-se que existem duas formas de votação principais. A primeira, votação projeto a projeto, é a forma mais frequentemente verificada nos orçamentos participativos. Por outro lado, alguns orçamentos participativos utilizam a denominada votação singular, onde o eleitor, mediante as regras, escolhe os projetos onde deseja exercer o seu voto e submete o mesmo ao servidor eleitoral. Apesar de esta ser a forma menos utilizada, verificou-se que 70% dos eleitores exercem os seus votos num período inferior a 5 minutos. Num período de 30 minutos este valor sobe para 84%. Esta é uma conclusão importante, pois já se tinha vindo a verificar que, na grande maioria dos casos, a votação singular, é aquela que mais se adequa.

Capítulo 6

Especificação da solução

6.1 Análise de requisitos

O objetivo principal desta dissertação é a criação e implementação de uma *RESTful API* para gestão de processos eleitorais de orçamentos participativos, geridos pela entidade acolhedora, que garanta os principais requisitos das votações por meios eletrónicos remotos. Desta forma, pretende-se elevar o grau de confiança nas soluções fornecidas pela entidade acolhedora no âmbito dos orçamentos participativos. Definimos como requisitos principais do sistema de votação eletrónica remota, no âmbito dos processos eleitorais dos orçamentos participativos, os seguintes: **Unicidade**, **Anonimato**, **Autenticidade**, **Auditabilidade**, **Disponibilidade** e a **Integridade**. Tendo em conta estes requisitos e o funcionamento dos processos eleitorais em orçamentos participativos, definiram-se os seguintes requisitos funcionais e não funcionais:

6.1.1 Funcionais

Para a definição dos requisitos funcionais ir-se-á utilizar a técnica MoSCoW. Esta técnica permite a classificação de requisitos nas seguintes quatro categorias:

- ***Must*** - Estes requisitos são **essenciais** para o sistema. Sem eles o sistema não pode ser dado como completo.
- ***Should*** - Estes requisitos são **importantes** para o sistema. Todavia, não são essenciais para a entrada em produção nesta fase. Por norma, estes requisitos não são tão vitais quanto os *Must* mas têm uma considerável importância.
- ***Could*** - Estes requisitos são **desejáveis** para o sistema pois podem enriquecer-lo e aumentar o seu valor. Todavia, estes não são parte do sistema base e, geralmente, são implementados quando existe tempo e recursos para tal. Alguns destes requisitos podem, inclusive, até nunca chegar a ser implementados.
- ***Won't*** - Estes requisitos apresentam uma **importância baixa e não compensam o investimento** de tempo, energia e orçamento. Eventualmente poderão ser considerados no futuro mas, atualemente, não apresentam benefícios significativos para o sistema.

A referida técnica apresenta uma grande desvantagem. Esta não ajuda a decidir entre vários requisitos dentro da mesma categoria. Qual implementar primeiro? De forma a dar

resposta a esta questão ir-se-á adicionar o atributo prioridade a cada um dos requisitos. Essa prioridade será classificada como **Baixa**, **Média** ou **Alta**. De seguida, na tabela 6.2 apresentam-se os requisitos funcionais identificados:

Requisito	**Breve explicação**	**Classificação**	**Prioridade**
R1 - Criar eleições	Deve ser possível registar uma nova eleição indicando um conjunto de parâmetros (ex: título, data início, data fim,...).	Must	Alta
R2 - Editar eleição	Deve ser possível editar uma eleição, criada por este cliente, enquanto a eleição não estiver congelada.	Must	Alta
R3 - Eliminar eleição	Deve ser possível poder eliminar uma eleição, criada por este cliente, enquanto a eleição não estiver congelada.	Must	Alta
R4 - Obter lista eleições	Deve ser possível obter uma lista das suas eleições.	Must	Alta
R5 - Detalhe eleição	Deve ser possível obter os detalhes de uma eleição sua.	Must	Alta
R6 - Registar eleitor	Deve ser possível registar um novo eleitor, numa eleição criada por si.	Must	Alta
R7 - Eliminar eleitor	Deve ser possível eliminar eleitores sob determinadas condições.	Must	Alta
R8 - Registar *trustee*	Deve ser possível registar *trustees* numa eleição criada por estes enquanto a eleição não estiver congelada.	Must	Média
R9 - Eliminar *trustee*	Deve ser possível poder eliminar um *trustee* de uma eleição criada por ele.	Must	Média
R10 - Obter lista *trustees*	Deve ser possível poder obter uma lista dos *trustees* de uma eleição criada por ele.	Must	Média
R11 - Congelar eleição	Deve ser possível congelar uma eleição criada por este. Somente após uma eleição estar congelada e estar dentro do período de votação é que os eleitores podem votar.	Must	Alta
R12 - Estender período de votação	Deve ser possível poder estender o período de votação de uma eleição, criada por ele, mesmo após o congelamento de uma eleição.	Should	Média

R13 - Arquivar eleição	Deve ser possível arquivar uma eleição criada por ele.	Should	Baixa
R14 - Adicionar voto encriptado	Deve ser possível adicionar votos encriptados a uma eleição.	Must	Alta
R15 - Adicionar voto	Deve ser possível adicionar votos, sem estarem encriptados, a uma eleição. O servidor será responsável por encriptar estes votos.	Must	Alta
R16 - Detalhe voto	Deve ser possível obter os detalhes de um voto.	Must	Alta
R17 - Adicionar voto a quarentena	Possibilidade de adicionar votos à quarentena para ser analisado.	Won't	Baixa
R18 - Retirar voto de quarentena	Possibilidade de um voto ser retirado da quarentena.	Won't	Baixa
R19 - Obter lista votos	Deve ser possível obter uma lista paginada com os votos (encriptados) de uma determinada eleição.	Must	Média
R20 - Obter recibo do voto	Deve ser possível obter um recibo do voto, submetido por um eleitor, assinado digitalmente pelo servidor.	Won't	Baixa
R21 - Contabilizar os votos	Após o término do período de votação deve ser possível proceder à contabilização dos votos.	Must	Alta
R22 - Submeter os *decryption factors* e as *decryption proofs* dos *trustees*	O processo de contabilização dos votos retorna o resultado da eleição encriptado. Para o desencriptar torna-se necessário que cada *trustee* desencripte parcialmente este resultado encriptado com a sua chave privada e que submeta os *decryption factors* e as *decryption proofs* ao servidor.	Must	Alta
R23 - Combinar os *decryption factors* dos trustees	Uma vez submetidos, e verificados, todos os *decryption factors* dos *trustees* estão reunidas as condições para se desencriptar o resultado da eleição. Deve ser possível a um cliente da API proceder à combinação e consequente desencriptação do resultado de uma eleição.	Must	Alta

R24 - Submeter voto auditado	Após a publicação dos resultados um votante pode submeter o voto que auditou.	Could	Alta
R25 - Estatísticas	Deve ser possível obter estatísticas (como o número de votos por dia, número de eleitores, total de votos, etc).	Won't	Baixa

Tabela 6.2: Requisitos funcionais

6.1.2 Não funcionais

- A solução deve permitir um número arbitrário de eleições, inclusive a decorrerem ao mesmo tempo.
- A solução deverá equacionar a existência de múltiplas e variadas regras de votação. O servidor de voto deve validar essas mesmas regras e garantir, independentemente destas, a segurança, a privacidade e os requisitos definidos anteriormente para os sistemas de votação eletrónica remota.
- A solução deve garantir o cumprimento dos requisitos de votação anteriormente referidos. Note-se que o requisito da não coercibilidade não está aqui equacionada uma vez que se entende que o risco de coerção neste tipo de votação não é elevado.
- Durante o desenvolvimento da solução, deve ser utilizado o sistema de controlo de versões GIT.
- A solução deverá executar sobre *containers* Docker.
- A solução deverá ser facilmente escalável.
- A solução deve utilizar um ou mais dos seguintes SGBD: MongoDB, MySQL e PostgreSQL.
- A solução desenvolvida utilizando o modelo arquitetural Representational State Transfer (REST).
- O processo de efetivação de um voto não deve demorar mais do que 5 segundos.

6.2 Enquadramento

6.2.1 Principais riscos aplicacionais

Os sistemas de votação eletrónica remota baseados na Web partilham um conjunto de riscos aplicacionais com outras aplicações para a Web. Segundo a OWASP [1], os 10 principais riscos que estas aplicações enfrentam são os seguintes:

- **A1:2017 - Injeção - Falhas de injeção**, por SQL ou NoSQL, podem fazer com que os interpretadores executem comandos não pretendidos e/ou acedam a informação não autorizada.

[1]Entidade sem fins lucrativos e com reconhecimento internacional, que atua com foco na colaboração para o fortalecimento da segurança de softwares em todo o mundo.

- **A2:2017 - Falhas de autenticação** - Falhas na implementação de autenticação e/ou gestão de sessões podem fazer com que um atacante comprometa *tokens* de autenticação, palavras-passe ou até mesmo permitir ao atacante assumir um determinado utilizador.
- **A3:2017 - Exposição de dados sensíveis** - Falhas na proteção de dados sensíveis podem levar ao comprometimento da confidencialidade que poderão lesar, financeiramente ou não, os utilizadores e/ou empresa. Proteções como a encriptação de dados, são geralmente adotadas para proteger esses mesmos dados.
- **A4:2017 - Entidades externas de XML (XXE)** - Alguns processadores de XML avaliam entidades externas que podem ser utilizados para revelar documentos internos que podem permitir vários ataques.
- **A5:2017 - Falhas no controlo de acessos** - Restrições sobre o que os utilizadores autenticados estão autorizados a fazer, nem sempre são corretamente verificadas o que pode permitir a um atacante aceder a dados e informações a que este não têm autorização.
- **A6:2017 - Configurações de segurança incorretas** - Más configurações são algo bastante comum em aplicações *web*. Um atacante poderá utilizar estas más configurações a seu favor e obter acesso a dados, sistemas e demais sem a devida autorização.
- **A7:2017 - Cross-site scripting (XSS)** - As falhas de XSS ocorrem sempre que uma aplicação inclui dados não confiáveis numa página *web* sem validação ou filtragem apropriada. Desta forma, um atacante pode executar *scripts* no *browser* com o intuíto de sequestrar sessões, descaracterizar a plataforma ou redirecionar o utilizador para *websites* maliciosos.
- **A8:2017 - Desserialização insegura** - "Desserializar" significa recuperar dados (ou estado de um objeto) a partir de um conjunto de bytes. Se este processo não for realizado com segurança, poderá permitir a um atacante executar código remotamente o que poderá ter sérias consequências.
- **A9:2017 - Utilização de componentes vulneráveis** - A utilização de componentes vulneráveis é um grande problema na medida em que deixa "portas" abertas para a exploração de vulnerabilidades. Dependendo da vulnerabilidade, o atacante pode obter acesso ao sistema, elevar os seus privilégios e obter informações confidenciais.
- **A10:2017 - Registo e monitorização insuficiente** - Registo e monitorização de atividades e alertas são importantes para dar resposta a ataques. Na falta deles, os atacantes conseguem explorar a aplicação sem sequer serem detetados em tempo útil para uma resposta atempada.

Dos riscos identificados alguns são mais prováveis de acontecer do que outros. De acordo com a experiência que a empresa possui no desenvolvimento de aplicações *web*, foi possível determinar que, em especial nos orçamentos participativos, tentativas de exploração de **XSS, *SQL e NoSQL injection*** e configurações de segurança incorretas, foram aquelas que representaram o maior número de tentativas de exploração.

Por consequência, os riscos suprareferidos podem levar ao comprometimento de um processo eleitoral. Para dar resposta e mitigar esses riscos, durante o desenvolvimento da *RESTfull API*, ir-se-á seguir as indicações/recomendações que constam no OWASP *Secure Coding Practices*.

6.2.2 Arquiteturas *single tenant* e *multi tenant*

Single Tenant

A arquitetura ***single tenant*** segue o princípio de alocar uma instância do servidor e da base de dados por cliente. Desta forma, cada cliente possui uma base de dados e instâncias aplicacionais exclusivas [1]. Essencialmente, nestas arquiteturas não existe partilha de recursos entre clientes.

Vantagens da arquitetura *single tenant*

- Bases de dados isoladas.
- Performance mais previsível [2]. A performance de um cliente não é afetada por algum processo de um outro cliente.
- Possibilidade de ter diferentes versões adaptadas às necessidades específicas do cliente. Por outro lado, este pode também representar uma dificuldade acrescida para a empresa à medida que o número de instâncias personalizadas ao cliente cresce.
- Fácil de implementar.

Multi Tenant

Por outro lado, a arquitetura ***multi tenant***, muito utilizada no modelo de *Software as a Service (SaaS)* (*software* como um serviço), segue um princípio diferente que é o de partilhar instâncias aplicacionais e/ou bases de dados entre todos os clientes [1, 4, 5]. Todavia, os dados de cada cliente são isolados, ou seja, estes não são acessíveis por outros clientes.

Nas arquiteturas *multi tenant* encontramos, geralmente, duas abordagens para o armazenamento dos dados. São elas:

- Utilizando apenas **uma base de dados para todos os clientes** [1]. Os registos são armazenados com o identificador único do cliente. Desta forma, a aplicação pode sempre apresentar aos clientes os seus, e apenas os seus, dados. Esta abordagem é muito simples de implementar, todavia, as aplicações devem adotar mecanismos para prevenir que dados de outros clientes possam ser mostrados a outros clientes ou a partes não autorizadas. Como os dados de todos os clientes estão numa base de dados, o risco de compromisso e a sua extensão é consideravelmente maior.
- Utilizando **uma base de dados por cliente** [1]. Os registos são armazenados em bases de dados distintas. Esta abordagem apresenta como grande vantagem um maior isolamento dos dados. Por outro lado esta abordagem não escala bem quando o número de clientes é elevado. Para aplicações com apenas alguns clientes ou onde seja necessário garantir um melhor isolamento dos dados, esta poderá ser uma abordagem a ter em consideração. Se, por outro lado, o número de clientes for elevado, a abordagem anterior poderá apresentar-se como a ideal. Todavia, uma análise deve ser feita caso a caso para perceber a melhor abordagem para o cenário concreto.

Vantagens da arquitetura *multi tenant*

- Redução de custos uma vez que os recursos são partilhados entre os clientes [5].

- Distribuição de novas versões ou correção de problemas mais fácil e rápida [5].
- Gestão centralizada.
- Maior facilidade e rapidez no fornecimento de versões de demonstração uma vez que não existe necessidade de instânciar novos servidores e bases de dados para estes.
- Redução na complexidade do processo de gestão do software para múltiplos clientes [4].

Na figura 6.1, que se apresenta de seguida, podem ver-se as diferenças entre a arquitetura *single tenant* e *multi tenant*.

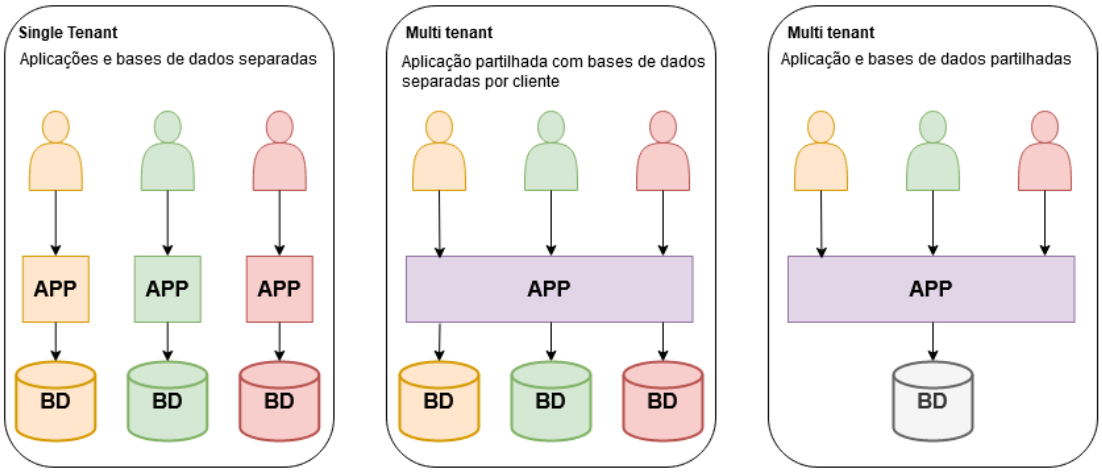

Figura 6.1: Arquiteturas *single* e *multi tenant*

6.2.3 Disponibilidade de serviço

A disponibilidade de serviço é muito importante em vários setores. Nos orçamentos participativos essa importância também se verifica. Durante uma eleição é muito importante que o serviço tenha uma disponibilidade elevada (idealmente 99.999999% do tempo). Todavia, manter este nível de serviço requer uma infraestrutura escalável, segura e eficiente. No entanto estes níveis não dependem somente da arquitetura mas também das aplicações sobre essa mesma arquitetura.

Replicação

Uma das formas de se garantir alta disponibilidade é através de replicação, de aplicações, servidores, *hardware* de rede e/ou *containers*[2].

De forma a fornecer um elevado nível de disponibilidade nos orçamentos participativos, em especial nos momentos de eleição, recomenda-se a replicação física e não física. Fisicamente podemos ter vários servidores e recorrer a balanceadores de carga para distribuírem os pedidos pelos vários servidores. Cada servidor pode conter uma ou mais instâncias da aplicação, por exemplo recorrendo a *containers*, para se ter ainda um maior nível de disponibilidade.

[2]Os containers proporcionam uma maneira padrão de empacotar código, configurações e dependências de uma aplicação num único objeto. Estes compartilham o sistema operacional instalado no servidor e são executados como processos isolados de recursos.

Uma boa solução, que permite exatamente o suprareferido e com elevada capacidade de escalabilidade, é a utilização de um *cluster kubernets*, que nada mais é do que um orquestrador de *containers* dentro de um *cluster* de máquinas. No entanto, outras soluções mais simples poderão eventualmente ser adotadas em função do orçamento de implementação e manutenção.

Monitorização e Escalonamento

Quando se pretende alcançar elevada disponibilidade de serviço é crucial que existam ferramentas de monitorização das aplicações e da própria infra-estrutura. Através da monitorização e análise dos dados, é possível redimensionar um cluster para dar resposta a um elevado nível de afluência ou, por outro lado, reduzir o número de nós computacionais se existir um decréscimo considerável do número de visitantes. Vários fornecedores de serviços na *cloud* fornecem hoje estas funcionalidades a valores acessíveis. Por outro lado, ao invés de ser necessário redimensionar o próprio *cluster*, poderá haver necessidade de escalar apenas as instâncias aplicacionais. Estando a solução assente sobre o *kubernets* escalar as instâncias aplicações pode ser feito com relativa facilidade e rapidez.

Acordo de nível de serviço

Quando a alta disponibilidade é de facto um objetivo estritamente necessário, pode ser necessário realizarem-se acordos sobre os níveis de serviço que o contratante espera do fornecedor. Estes níveis de serviço devem ser explícitos e mensuráveis. Geralmente, quando existe uma falha nesse acordo, poderá haver lugar a indemnizações.

Atualmente, os maiores *players* do mercado dos serviços sobre infraestruturas *cloud*, estão capacitados para realizar este tipo de acordos para clientes que os necessitem. Por vezes estes cobram valores superiores para conseguirem fornecer um melhor serviço. A título de exemplo, a *Digital Ocean* [3] alega que as suas *droplets* [4] têm 99.99% de disponibilidade e que não cobra pelo tempo que estas estiverem indisponíveis.

Proteção face a ataques de negação de serviço

O Helios Voting não endereça, nem pretende endereçar, este problema uma vez que este sai do âmbito do mesmo. No entanto, considerando que é um problema atual e tendo em conta a existência de serviços *online* que, por um valor muito baixo, disponibilizam *botnets* [5] para levar a cabo estes ataques, decidiu-se que seria prudente equacionar a sua ocorrência e prover a infraestrutura de mecanismos de defesa contra estes ataques.

Para colmatar as ameaças à disponibilidade do serviço, e uma vez que a infraestrutura está alojada na Cloud, recomenda-se a utilização de um serviço especializado. A análise realizada apontou o Cloudflare como uma possível solução. Após análise dos serviços disponibilizados pela Cloudflare, constatou-se, de facto, que os mesmos têm uma infraestrutura capaz de monitorizar, detetar, alertar e reagir a estes ataques. Estas características combinadas com a sua gigante infraestrutura, com capacidade, segundo a própria Cloudflare, para mais de 37 Tbps e um custo muito reduzido, fazem com que este serviço seja o "parceiro" ideal na proteção dos servidores dos orçamentos participativos (mas também de outros servidores aplicacionais da entidade empregadora).

Uma vez decidido que a *RESTful API*, assim como os servidores eleitorais não estarão expostos diretamente na internet, proteger-se-ão os servidores dos orçamentos participativos

[3] Fornecedor americano de infraestruturas em nuvem.

[4] Servidores virtuais que podem ser instanciados em alguns segundos.

[5] Rede de computadores que foram infectados por *softwares* maliciosos e que podem ser controlados remotamente para enviar *spam*, espalhar vírus ou executar ataques de negação de serviço (DDOS) sem o conhecimento ou o consentimento dos seus donos.

protegendo, consequentemente, os servidores eleitorais. Assim sendo, torna-se necessário que todas as aplicações expostas para a internet sejam protegidas pela Cloudflare.

6.2.4 Autenticação

Nesta subsecção pretendem-se apresentar alguns métodos de autenticação que foram analisados e considerados para a implementação da autenticação. Começaremos abordando os métodos tradicionais baseados em sessões, seguiremos para os baseados em *tokens* e, por fim, abordaremos os métodos baseados em chaves de API.

- **Baseada em sessões** - Na autenticação baseada em sessões, o servidor cria uma sessão para o utilizador assim que este insere corretamente as credenciais de acesso. Esta sessão é guardada na memória do servidor durante um determinado período de tempo. O identificador de sessão gerado é então armazenado como *cookie* no *browser* do cliente. Como os *cookies* são enviadas em todos os pedidos, o servidor pode então verificar, confrontando a sua tabela de sessões, a identidade do utilizador e se a sessão ainda está válida. Um exemplo do referido processo pode ser visto na figura 6.2.

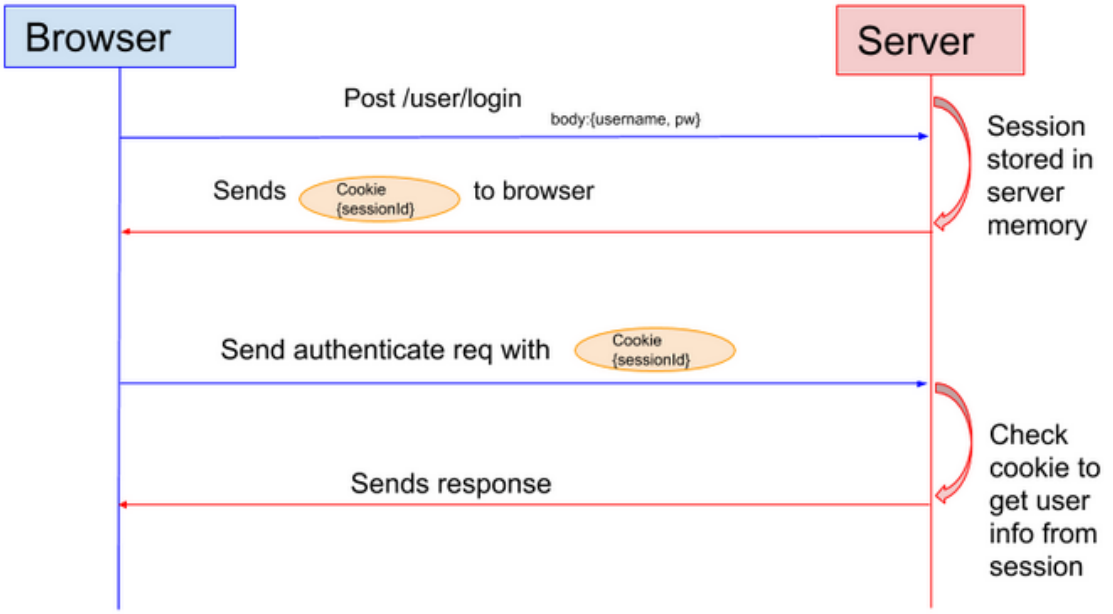

Figura 6.2: Exemplo de autenticação baseada em sessões [23]

- **Baseada em *tokens*** - Na autenticação baseada em *tokens*, o servidor cria um *token* assim que são inseridas corretamente as credenciais de acesso. Este *token* é então enviado para o *browser* do cliente que o guarda, geralmente em *localstorage*, e envia-o a cada pedido que é feito ao servidor que ao receber o pedido valida esse mesmo *token*. Ao contrário do que acontece em autenticações baseadas em sessões, o servidor não armazena qualquer informação em memória pois o *token* em si tem informação que permite ao servidor autenticar o cliente. Os *tokens* emitidos pelo servidor são sempre assinados, com uma chave secreta, para garantir que o *token* foi emitido pelo servidor e não foi alterado ou criado por uma qualquer terceira parte. Um exemplo do referido processo pode ser visto na figura 6.3.

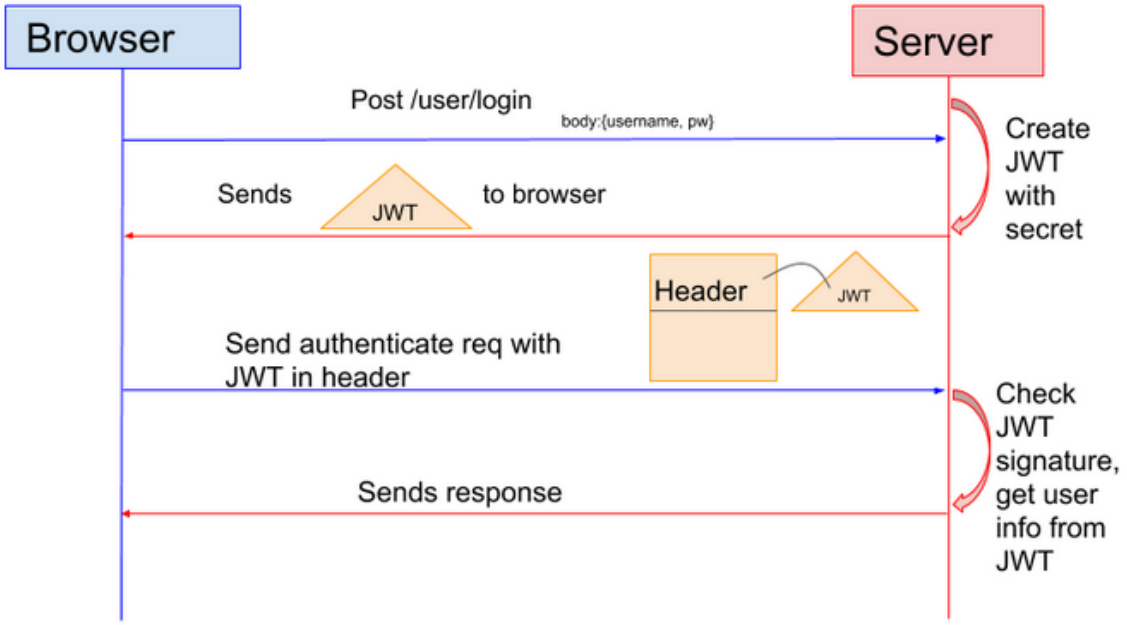

Figura 6.3: Exemplo de autenticação baseada em *tokens* [23]

Neste momento o leitor poderá estar a debater-se com a seguinte questão: porquê utilizar este tipo de autenticação? A resposta a esta questão, mencionando apenas algumas vantagens, é a seguinte:

- os *tokens* são ***stateless***, isto é, eles contém todas as informações necessárias para que o servidor possa autenticar o cliente. Desta forma não existe nenhuma necessidade de se armazenar em memória o estado da sessão.
- os *tokens* **podem ser gerados em qualquer lugar**. A geração de *tokens* é dissociada da verificação dos *tokens*. Desta forma a sua geração pode ser feita em servidores separados ou mesmo através de empresas que forneçam serviços de autenticação, como a Auth0.

Hoje em dia muitos serviços e aplicações para a *web* utilizam JSON Web Token (JWT), um standard aberto (RFC 7519) que define um método compacto e independente para transmitir com segurança informações entre partes codificadas como um objeto JSON. O JWT ganhou grande popularidade devido ao seu tamanho compacto, que permite que os *tokens* sejam facilmente transmitidos por meio de strings de consulta, atributos de cabeçalho de um pedido ou dentro do corpo de uma solicitação POST. Atualmente, a autenticação através do JWT é um dos métodos mais utilizados para autenticação nas plataformas da entidade acolhedora.

- **Baseada em chaves de API (API KEYS)** - Na autenticação baseada em chaves de API é gerada uma chave, exclusiva, por cada cliente. Por norma, esta chave é mantida associada a um cliente. Quando o servidor recebe um pedido, com uma chave, irá verificar se esta é uma chave válida. Este tipo de autenticação faz apenas sentido se:
  - **forem utilizados canais de comunicação seguros** uma vez que o comprometimento da chave de API permitiria a uma entidade não autorizada executar ações perante um sistema como se fosse o próprio cliente. **OU**

– **a informação transmitida estiver contida numa rede privada** isolada. Todavia, mesmo nestas, se possível, devem ser utilizados canais de comunicação seguros.

Um exemplo do funcionamento deste método pode ser visto na figura 6.4.

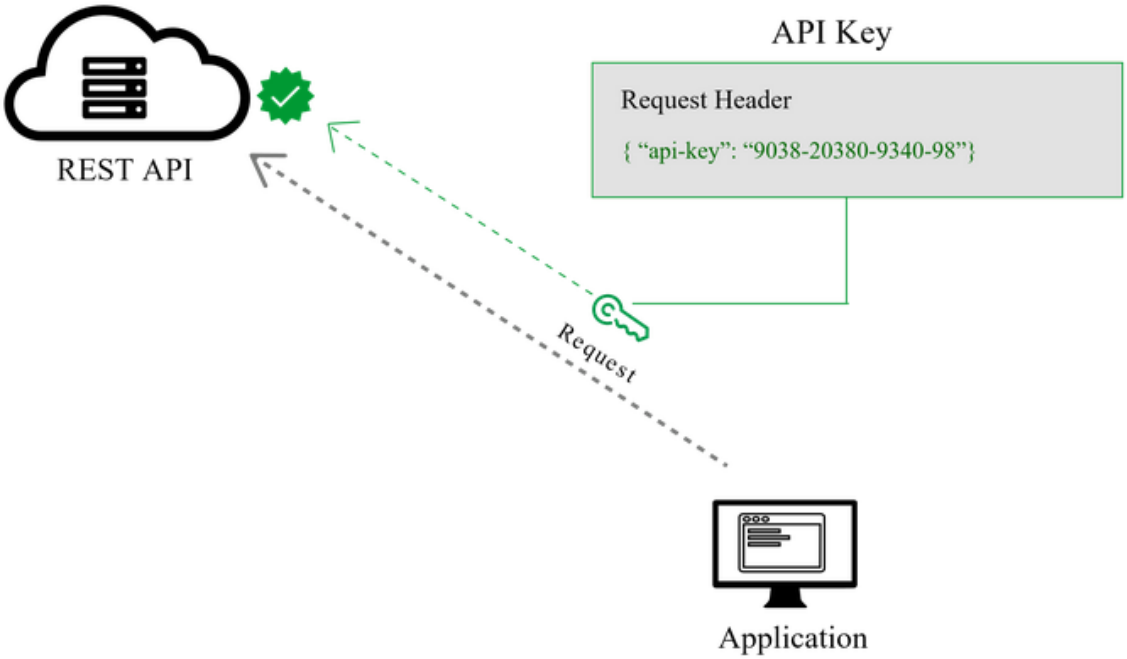

Figura 6.4: Exemplo de autenticação por chave de *API* [30]

6.3 Arquitetura base

Tendo em conta os principais riscos, os requisitos que se pretendem garantir e a análise das várias propostas de sistemas de votação eletrónica, decidiu-se que a implementação da *RESTful API* terá por base o **Helios Voting**, proposto por Ben Adida. Tal decisão, teve em conta não só o facto de o Helios garantir o cumprimento dos requisitos pretendidos, mas também pelo seguinte motivos:

(a) não exigir de qualquer equipamento específico;

(b) não exigir interações complexas aos eleitores;

(c) não exigir uma infraestrutura com elevado custo monetário nem de manutenção;

(d) permitir que os eleitores verifiquem que o seu voto foi inserido, interpretado e contabilizado corretamente pelo sistema eleitoral;

(e) permitir que qualquer entidade possa auditar o processo eleitoral.

Uma das questões que surgiram desde o início era sobre a necessidade do servidor eleitoral, que expõe a *RESTful API*, estar exposto diretamente na internet. Após discussão, decidiu-se que não ficará exposta diretamente. Assim sendo, decidiu-se que os servidores dos orçamentos participativos serão responsáveis por mediar todas as comunicações relativas a um processo eleitoral. Outras soluções, como por exemplo a utilização de uma *api*

gateway, foram equacionadas. No entanto, as mesmas comportavam a necessidade de uma infraestrutura mais complexa e monetariamente mais dispendiosa sem trazer grandes benefícios.

Arquitetura

Tendo em conta o referido ao longo deste capítulo apresenta-se de seguida, na figura 6.5, a arquitetura proposta.

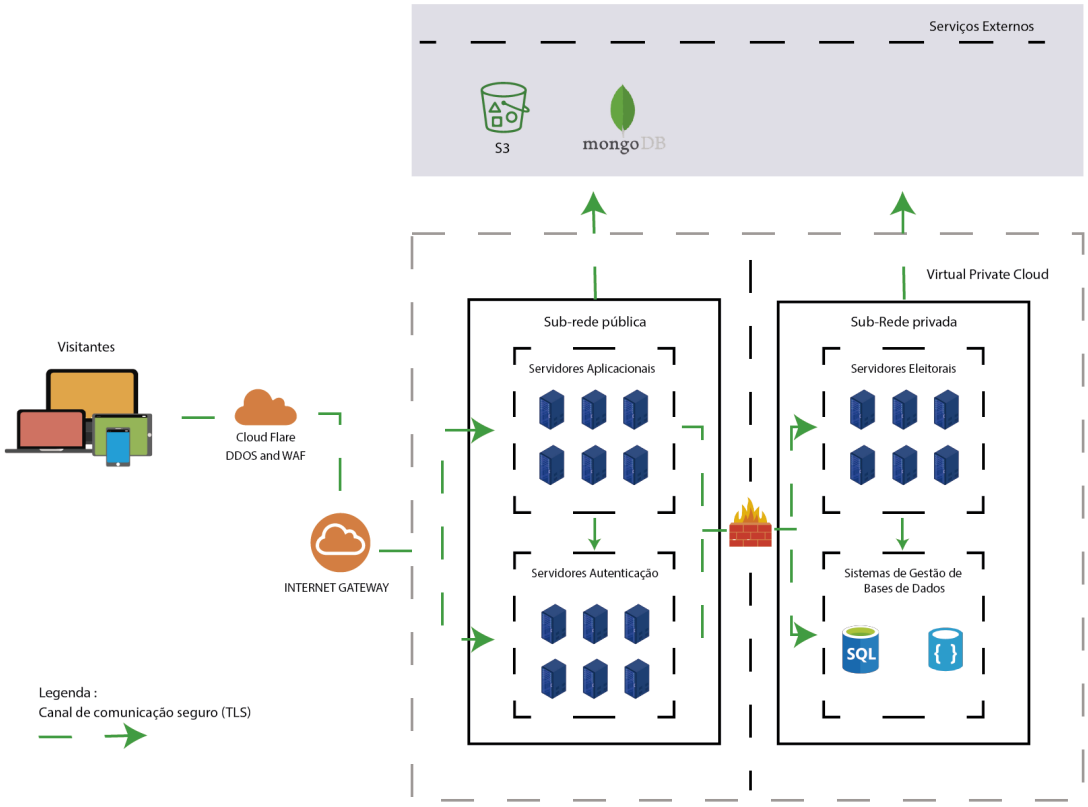

Figura 6.5: Arquitetura proposta

Como se pode constatar através da figura 6.5, os visitantes acedem aos servidores aplicacionais e de autenticação por intermédio da Cloudflare e de uma *Internet Gateway*. A Cloudflare, como vimos, desempenha um papel bastante importante que é proteger a sub-rede pública de ataques de negação de serviço (inclusive distribuídos), e proteção dos servidores aplicacionais através de uma *Web Application Firewall (WAF)* face aos principais riscos aplicacionais referidos anteriormente.

Nesta proposta optou-se por segmentar os servidores em duas sub-redes distintas:

- a **sub-rede pública**, na qual constam os servidores aplicacionais (dos orçamentos participativos) e os servidores de autenticação e autorização expostos através da *Internet Gateway*.
- a **sub-rede privada**, na qual constam sistemas de gestão de bases de dados relacionais e não relacionais assim como os servidores eleitorais replicados. As comunicações com esta sub-rede são controladas por *firewall* que deverá permitir apenas comunicações que tenham origem dentro da *Virtual Private Cloud*. Outras regras poderão ser adicionadas para um controlo mais refinado das comunicações entre servidores.

Nesta proposta, os servidores eleitorais não estão expostos para a internet como referido anteriormente. Tal decisão prende-se essencialmente com motivos de segurança mas também com o facto de a entidade acolhedora pretender separar responsabilidades aplicacionais e gerir todos os processos eleitorais através de uma arquitetura *multi tenant* (multi-cliente) facilmente escalável e suficientemente abrangente em termos de regras que podem ser implementadas.

Tendo em conta o referido, os servidores dos orçamentos participativos serão responsáveis por interagir com os servidores eleitorais, através de pedidos autenticados, de forma a gerir todo o processo eleitoral. O diagrama de sequência na figura 6.6 representa, de forma generalizada, as interações entre um utilizador, o servidor do orçamento participativo e o servidor eleitoral. A autenticação dos eleitores é também uma responsabilidade dos servidores dos orçamentos participativos.

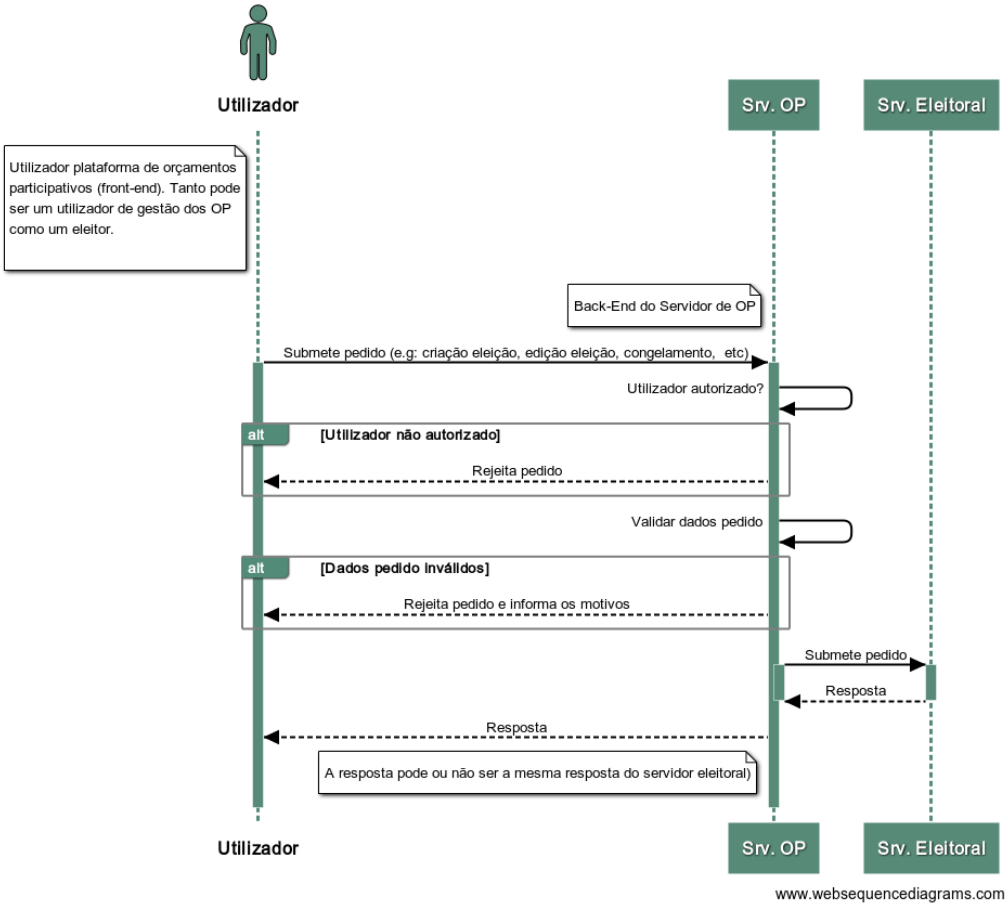

Figura 6.6: Diagrama sequência generalizado das interações Utilizador, Servidor orçamento participativo e o Servidor eleitoral

6.3.1 Atores

Umas das primeiras questões que se colocaram aquando o início da presente proposta, era sobre quais os atores que iriam ter um papel no sistema que se pretende implementar. Da análise efetuada concluiu-se que, nesta proposta, fazem sentido os seguintes atores: **administradores** e **gestores eleitorais** (os servidores dos orçamentos participativos). Os **administradores** terão, como papel principal, a gestão dos clientes e outros administradores que poderão utilizar o sistema. Por outro lado, os **gestores eleitorais** irão ter como

papel principal a gestão de processos eleitorais por eles criados.

De seguida apresentam-se dois casos de uso, um para cada ator, com algumas das funcionalidades que os mesmos podem executar no sistema.

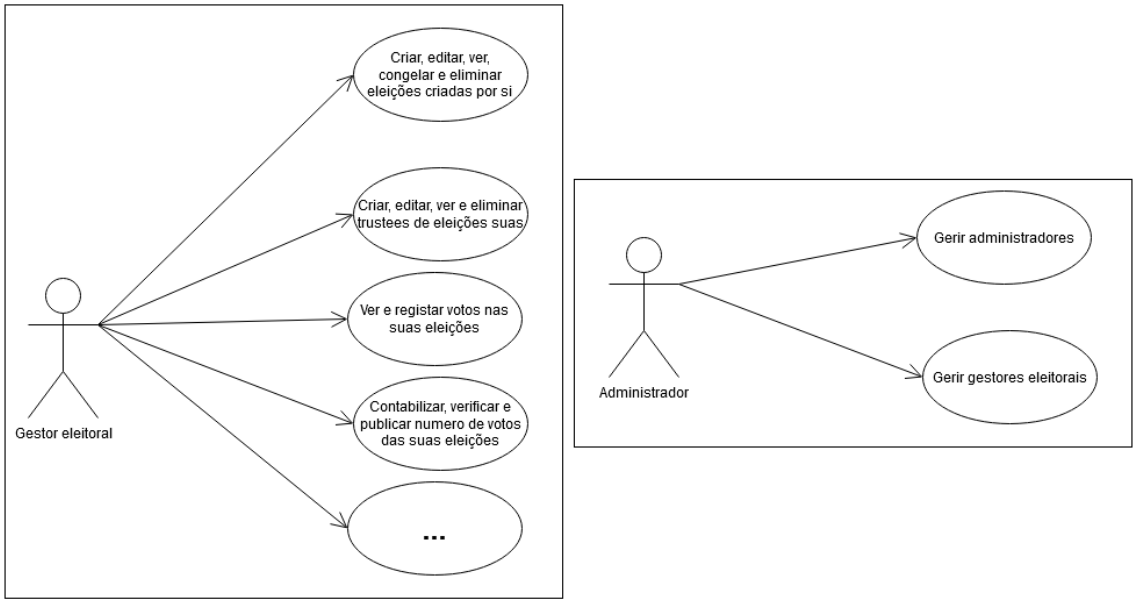

Figura 6.7: Diagrama de casos de uso de interações dos atores com o sistema

6.3.2 Linguagem de programação e *framework*

Como linguagem de programação ir-se-á utilizar o **Python** essencialmente porque toda a base do servidor Helios foi desenvolvida em Python2. Vários algoritmos, técnicas e métodos criptográficos especificamente desenvolvidos para o Helios e cujas implementações são bastante sensíveis estão codificados nesta linguagem. Para além do referido, não aparentam existir vantagens claras em reimplementar o Helios numa outra linguagem quando, esta implementação em Python, provou ao longo de vários anos ser bastante robusta. Todavia, uma vez que o suporte para o Python2 terminou ir-se-á adaptar a solução para a versão mais recente que é o Python3.

Para a codificação da *RESTful API* ir-se-á recorrer à ***Django Rest Framework***. Esta é uma *framework* assente na *framework* Django, específica para desenvolvimento de serviços *REST*, que facilita no desenvolvimento deste tipo de serviços.

6.3.3 Biblioteca Helios

Após a análise do código fonte do Helios decidiu-se que seria uma boa ideia transformar aquele código numa biblioteca Python. Esta biblioteca deveria utilizar o código original do Helios e utilizar o paradigma de programação orientada aos objetos. Uma vez decidido que este era o "caminho a seguir", começou-se a implementação da biblioteca. Todavia, durante as pesquisas efetuadas, foi possível encontrar uma biblioteca que implementava o que se pretendia e utilizando o código da proposta original. Assim sendo, verificou-se e testou-se o seu funcionamento e constatando-se que funcionava corretamente optou-se por utilizar essa mesma biblioteca de forma a acelerar o desenvolvimento.

A biblioteca que se decidiu utilizar pode ser encontrada através do seguinte url: `https:`

`//github.com/anarvote/helios_lib`. Ainda assim, esta biblioteca sofre de um mesmo problema verificado na implementação original do Helios que é a utilização de métodos de *hashing* inseguros, mais concretamente, a utilização de SHA1 (considerado inseguro desde 2009). Para colmatar este problema, clonou-se esta biblioteca e fizeram-se as alterações necessárias, também nos testes existentes, para alterar o método de *hashing* para SHA256 que é um método de *hashing* seguro ao dia de hoje.

6.4 Conclusão

O presente capítulo iniciou-se com a análise dos requisitos funcionais e não funcionais, essenciais ao sistema a implementar. Para os requisitos funcionais decidiu-se utilizar uma técnica bastante conhecida, técnica de MoSCoW, para a sua classificação. Para possibilitar uma priorização dos requisitos, dentro de cada categoria da MoSCoW, decidiu-se classificar os requisitos com as seguintes prioridades: baixa, média ou alta.

Posteriormente, identificaram-se os principais riscos aplicacionais aos quais os sistemas de votação electrónica remota estão expostos, introduziu-se o conceito de arquiteturas *single tenant* e *multi tenant* apresentando-se as vantagens e desvantagens de cada, reiterou-se a importância da disponibilidade de serviço e algumas formas de se conseguir obter níveis elevados de disponibilidade. E como a autenticação será claramente parte intrínseca do nosso sistema, procedeu-se também à análise de três métodos de autenticação frequentemente utilizados.

Por fim, propôs-se a implementação da arquitetura proposta por Ben Adida em 2009 para votação eletrónica remota explicando-se os principais motivos por detrás desta escolha. Neste contexto, apresentou-se ainda a separação lógica entre os servidores dos orçamentos participativos e os seus servidores de autenticação, e os servidores eleitorais e sistemas de gestão de bases de dados. Para proteger a infraestrutura de ataques de negação de serviço, propôs-se ainda o recurso aos serviços prestados pela CloudFlare. Apresentou-se também o diagrama de sequência generalizado das interações entre o Utilizador, o Servidor do orçamento participativo e o Servidor eleitoral. Para terminar o capítulo apresentaram-se também os atores, que irão ter um papel no sistema a implementar, a linguagem de programação e biblioteca à qual se recorrerá.

Capítulo 7

Implementação da *RESTful API*

Neste capítulo ir-se-á abordar a implementação da *RESTful API* proposta e que é a principal contribuição desta dissertação para a entidade acolhedora. Note-se que, neste capítulo, apenas serão apresentados alguns dos requisitos implementados. Os restantes poderão ser consultados nos anexos B, C, D, E, F, G, H e I.

7.1 Autenticação dos gestores eleitorais

Como é que os gestores eleitorais, servidores de orçamentos participativos ou outros, podem autenticar-se na *RESTful API* que se pretende implementar? Como é que, tendo os servidores replicados, os gestores eleitorais poderão ser autenticados independentemente da réplica para onde um determinado pedido seja encaminhado?

Após análise e discussão **decidiu-se pela autenticação por chave de API** pelos seguintes motivos:

- A *RESTful API* a implementar apenas irá estar disponível dentro de uma rede interna.
- A *RESTful API* a implementar apenas irá ser utilizada através de outras API's, no caso, API's dos servidores de orçamentos participativos.
- A *RESTful API* irá recorrer ao protocolo *Transport Layer Security (TLS)* para garantir a segurança dos dados em trânsito.
- A implementação de um sistema por chaves de API é de fácil implementação e não requer conhecimentos especializados pelo que pode ser facilmente gerida pelos elementos da equipa.

Nota: Todos os pedidos a esta Application programming interface (API) têm que, obrigatoriamente, ser autenticados. Sem credenciais válidas todos os pedidos serão rejeitados. De ora em diante iremos ter sempre pressuposta esta questão.

7.2 Alteração dos algoritmos de hashing do Helios

A análise ao código do Helios evidenciou a existência e utilização de um algoritmo de *hashing* considerado, desde 2009, como inseguro e, portanto, a sua utilização não é reco-

mendada. Tendo em conta o referido, decidiu-se validar a possibilidade de se utilizar um outro algoritmo de *hashing* que, aos dias atuais, seja considerado um algoritmo seguro. Assim sendo, procedeu-se à verificação do impacto da alteração e concluiu-se que a alteração é possível, desejada, técnicamente viável e relativamente fácil de adaptar.

Inicialmente, procedeu-se à identificação dos pontos, no código, onde se estava a utilizar o SHA1. De seguida, identificaram-se 7 pontos onde se teria que alterar o algoritmo de *hashing*. Note-se que a alteração do algoritmo de *hashing* utilizado no Helios comporta a necessidade de, obrigatoriamente, se alterar a biblioteca de *front-end* para utilizar o mesmo algoritmo que o *back-end*.

Algoritmo de *hashing* a utilizar

Para colmatar os problemas que o SHA1 apresenta tornou-se evidente que teria que se recorrer a um outro algoritmo. Todavia não era claro o algoritmo que poderia ser utilizado. Após várias pesquisas concluiu-se que o algoritmo *SHA256* se apresenta como o algoritmo recomendado, atualmente, para o efeito. Assim sendo, e verificando-se a exequibilidade e existência nativa da implementação desta função de *hashing* na biblioteca *hashlib*), do python, procedeu-se à alteração do algoritmo para o SHA256 nos pontos de seguida identificados.

Pontos identificados

- algs.py -> prove_decryption;
- algs.py -> EG_disjunctive_challenge_generator;
- algs.py -> DLog_challenge_generator;
- elgamal.py -> prove_decryption;
- elgamal.py -> EG_disjunctive_challenge_generator;
- elgamal.py -> DLog_challenge_generator;
- randpool.py -> __init __;
- utils.py -> hash_b64;

7.3 R1 - Criação de eleições

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.

Funcionamento:

As eleições são parte fundamental dos processos eleitorais e um requisito fundamental no contexto desta dissertação. O processo de criação de eleições é bastante simples e funciona da seguinte forma:

1. Ao receber os dados da eleição enviados pelo **cliente**, a API irá fazer a sua validação de acordo com as regras definidas. Se os dados recebidos não estiverem de acordo com as regras, o pedido é rejeitado e são retornados os motivos pela qual a eleição não pode ser criada.

2. Estando o pedido de acordo com as regras definidas é gerado um par de chaves criptográficas, através do algoritmo ElGamal, para o *trustee* do servidor eleitoral. Todas as eleições possuem, no mínimo, um *trustee* o qual se denomina de *Helios Trustee*.

3. De seguida, a eleição e o *trustee* do servidor eleitoral são criados e armazenados no SGBD através dos dados do pedido e par de chaves criptográficas geradas, respetivamente. É também gerado um *hash* criptográfico, utilizando o algoritmo SHA256, para a chave pública gerada. Note-se que chave privada do *trustee* não é, em nenhuma circunstância, revelada ao cliente através da API.

4. Por fim, são retornados os atributos, definidos explicitamente, da eleição criada.

Diagrama de sequência

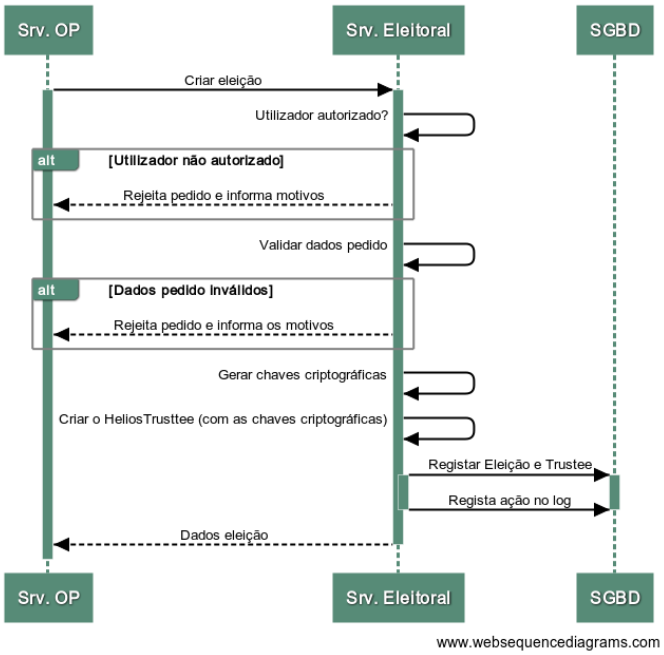

Figura 7.1: Diagrama sequência: Criação de eleição

7.4 R2 - Edição de eleição

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.

- A eleição não pode estar congelada (*freezed*).
- Par de chaves criptográficas da eleição, data de criação, cliente que criou a eleição (*tenant*), *freezed* e uuid **não são campos editáveis**. Quando enviados por um cliente no pedido estes devem ser descartados.
- O cliente só pode editar eleições criadas por si.

Funcionamento:

Através de um pedido PUT ou PATCH os clientes da API podem facilmente alterar determinados atributos enquanto a eleição não estiver congelada. O processo de edição de uma eleição funciona da seguinte forma:

1. Ao receber os dados a API irá descartar os atributos não editáveis e fazer a validação dos restantes dados de acordo com as regras definidas no modelo. Se algum dos dados não estiver de acordo com as regras, o pedido é rejeitado e são retornados os motivos pelo qual não foi possível proceder à edição, ou seja, que regras não são cumpridas.
2. De seguida, procede-se à atualização da eleição no SGBD. Caso a atualização seja bem sucedida, retornar-se-ão os dados atualizados da eleição. Caso contrário, uma mensagem de erro irá ser retornada.

Diagrama de sequência

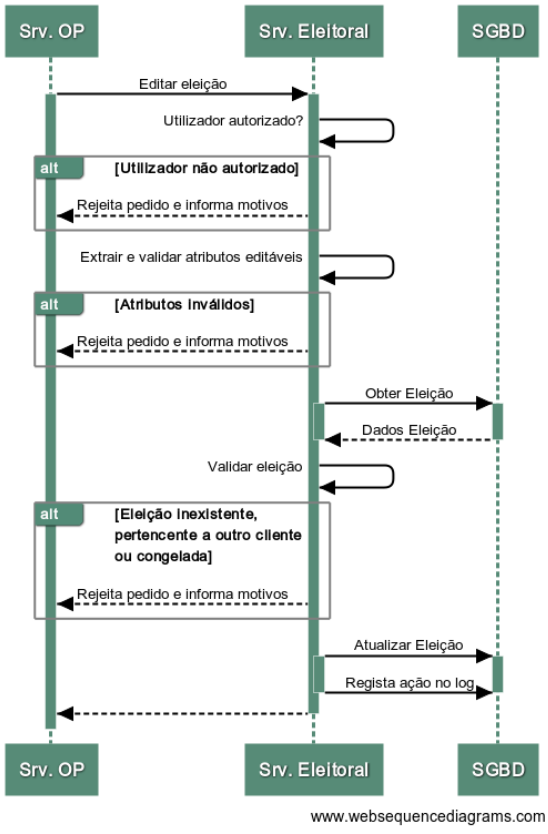

Figura 7.2: Diagrama sequência: Edição de eleição

7.5 R6 - Registar eleitor

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode adicionar eleitores em eleições criadas por si.

Funcionamento:

Através de um pedido POST os clientes da API podem registar os seus eleitores. A criação dos eleitores só pode ser feita se, e apenas se, forem cumpridas as seguintes condições:

- **Tratando-se de uma eleição aberta**, isto é, uma eleição que não tem um período de registo de eleitores pré-eleitoral, o registo pode ser feito **até ao término do período de votação**. OU
- Sendo uma **eleição restrita**, até ao **congelamento da eleição**.

Na proposta aqui apresentada, o registo dos eleitores é feito através das aplicações dos orçamentos participativos. Como tal, decidiu-se que estes deveriam facultar obrigatoriamente o nome (se a eleição estiver definida para usar *alias* este campo é opcional) e o identificador único do eleitor no orçamento participativo. Tendo estes dados o eleitor é registado e é retornada a resposta ao pedido.

Diagrama de sequência

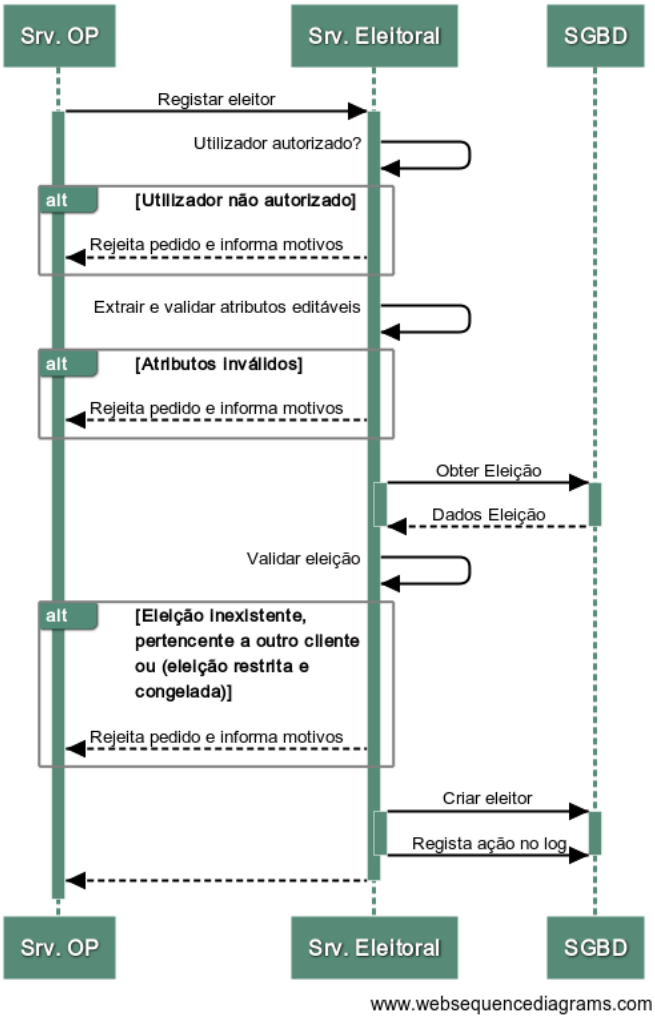

Figura 7.3: Diagrama sequência: Registo de eleitor

7.6 R8 - Registar *trustee*

Classificação: Must (Essencial)

Prioridade: Média

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode adicionar *trustees* eleitores a eleições criadas por si.
- A eleição não pode estar congelada.

Funcionamento:

Em eleições que necessitem de um elevado nível de segurança, a adição de um ou mais *trustees*, para além do *trustee* do servidor eleitoral (denominado de HeliosTrustee), como se referiu na secção 4.2, é adequada e recomendada. Para o registo de um *trustee* numa eleição, é necessário que seja facultada à API o nome do *trustee*, a sua chave pública (que deve ser gerada recorrendo ao algoritmo ElGamal) e também a prova de que este conhece a chave privada associada. Esta prova é de facto muito importante na medida em que, no momento do registo, existia conhecimento da chave privada associada à chave pública facultada ao servidor.

Tendo na posse estes dados, o *trustee* é então registado na base de dados. Se durante este registo ocorrer algum erro é retornada uma mensagem a informar a ocorrência do mesmo. Se, por outro lado, se obtiver sucesso no registo do *trustee*, na base de dados é retornada uma resposta de sucesso.

Diagrama de sequência

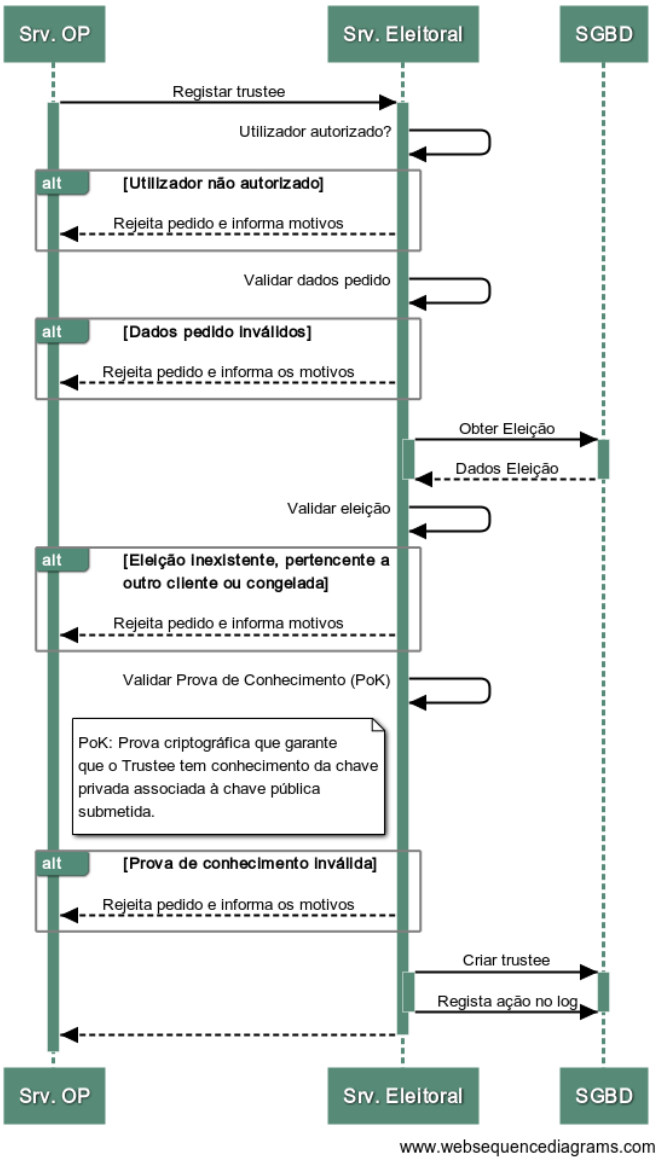

Figura 7.4: Diagrama sequência: Registo de *trustee*

7.7 R9 - Eliminar *trustee*

Classificação: Must (Essencial)

Prioridade: Média

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode eliminar um *trustee* de uma eleição criada por si.
- A eleição não pode estar congelada.

Funcionamento:

Através de um pedido DELETE, os clientes da API podem proceder à eliminação de um *trustee* desde que a eleição não esteja congelada.

Diagrama de sequência

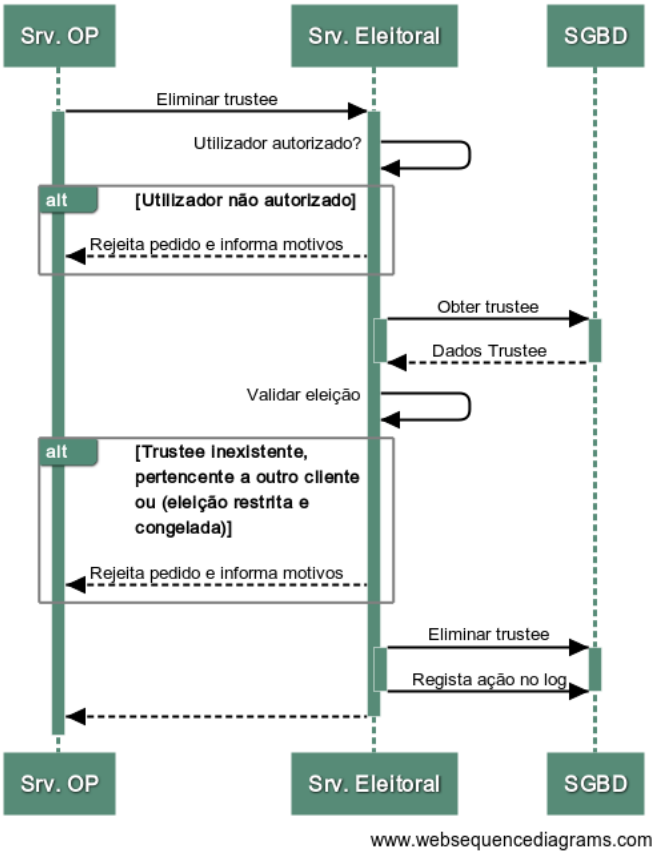

Figura 7.5: Diagrama sequência: Eliminação de um *trustee*

7.8 R11 - Congelar eleição

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode congelar uma eleição criada por si.

Funcionamento:

O congelamento de uma eleição é possível através de um pedido com o método PUT. Neste pedido deve ser facultado o identificador da eleição a congelar. Depois de verificado que a eleição a congelar foi criada pelo cliente autenticado, é verificado se esta já se encontra porventura congelada. Caso esta esteja já congelada o processo termina retornando uma mensagem com o status 403 a informar que a eleição já se encontra congelada. Caso contrário, procede-se da seguinte forma:

1. Se a eleição for restrita a um conjunto de eleitores definidos antes do período de votação, estes eleitores são obtidos e é criado um *array* de *dicts* [1] com os dados destes eleitores. Esse *array* é convertido numa string com o formato *JavaScript Object Notation (JSON)* para se gerar o *hash* dos eleitores.
2. Neste momento ir-se-á gerar o *hash* da eleição. Para tal, é criada uma estrutura, do tipo *dict*, com os dados já definidos da eleição. De seguida, esta estrutura é convertida numa string com o formato JSON recorrendo ao método *json.dumps* (nativo do Python). Por fim recorrer-se-á a uma função de *hashing* para gerar o *hash* da eleição.
3. Por fim, a eleição é atualizada definindo-se os seguintes atributos: **frozen_at** com a data e hora em que a eleição foi congelada, **voters_hash** com a *hash* dos eleitores (se votação restrita) e **election_hash** com a *hash* da eleição. Neste momento, são também combinadas as chaves públicas, através da operação de multiplicação, de todos os *trustees* da eleição e definido o atributo **public_key** com o resultado combinado das chaves públicas. Se a operação de definição destes atributos tiver sucesso, é retornada uma mensagem de sucesso. Caso contrário é retornada uma mensagem de erro.

Note-se que na proposta original do Helios apenas era equacionado o *hashing* dos eleitores da eleição quando a mesma era restrita. Todavia, nesta proposta, acreditamos que gerar um *hash* com os dados da eleição, poderá dar maior confiança aos eleitores pois desta forma podem perceber se existiu alguma alteração nos atributos da eleição.

[1] Um *dict* é uma estrutura de dados mutável, sem ordem e indexada representada na notação JSON. Exemplo: {"nome": "Leandro"}.

Diagrama de sequência

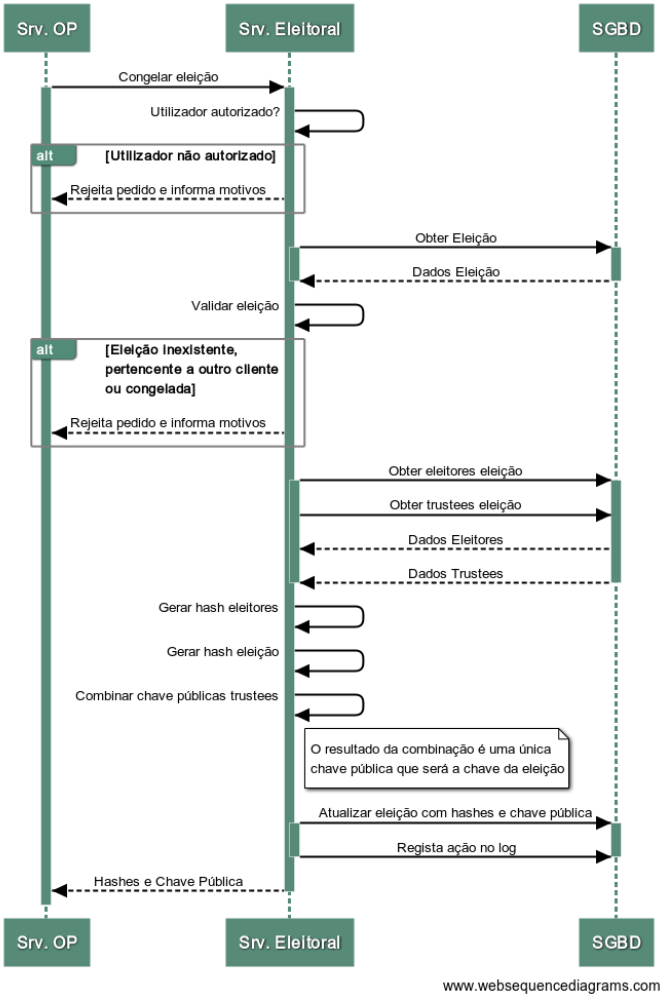

Figura 7.6: Diagrama sequência: Congelamento de uma Eleição

7.9 R14 - Adição de voto encriptado

Classificação: Must

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode adicionar votos em eleições criadas por si.
- Período de votação a decorrer.

Funcionamento:

A adição de um voto encriptado é possível através de um pedido com o método POST. No corpo do pedido deve ser facultado o voto encriptado, o *hash* da eleição e o identificador do eleitor. De seguida apresenta-se, resumidamente, o funcionamento deste processo:

1. Ao receber o pedido a API consulta a base de dados e obtém a eleição, na qual se pretende depositar o voto, e o eleitor que realizou o voto. Caso um destes dados não seja encontrado o pedido é rejeitado e são retornados os motivos pelo qual o pedido não pôde ser satisfeito.

2. Estando na posse da eleição, ir-se-á verificar se o *hash* da eleição enviado no corpo do pedido é igual ao *hash* da eleição atual. Esta verificação pretende garantir ao eleitor que o seu voto só é adicionado se a eleição não tiver sido alterada entretanto. Caso não sejam iguais o pedido é rejeitado e é também retornada uma mensagem a informar o motivo pelo qual o pedido não pôde ser satisfeito.

3. De seguida, proceder-se-á à verificação da corretude do boletim de voto encriptado. Neste processo, através de propriedades e técnicas criptográficas, pretende-se garantir que o boletim de voto foi preenchido de acordo com as regras da eleição. Caso as mesmas sejam violadas, o pedido é rejeitado e é retornado o motivo pelo qual não pôde ser satisfeito.

4. O próximo passo é gerar a "impressão digital" do voto através do algoritmo de *hashing* SHA256. Este *hash* é público e permite verificar se os dados do boletim de voto encriptado não foram alterados.

5. Finalmente, o boletim de voto encriptado é armazenado no SGBD. Dado que este boletim é encriptado, pode ser publicamente exposto (assumindo que as chaves criptográficas privadas dos *trustees* da eleição não sejam conhecidas).

Diagrama de sequência

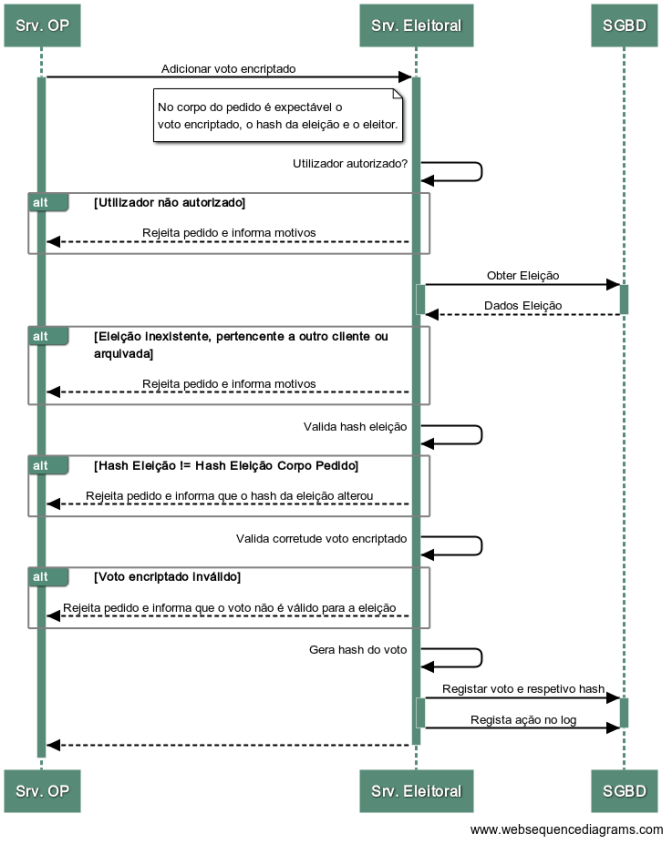

Figura 7.7: Diagrama sequência: Adição de um voto encriptado

7.10 R15 - Adição de voto

Classificação: Must

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode adicionar votos em eleições criadas por si.
- Período de votação a decorrer.

Funcionamento:

O funcionamento deste processo é em todo semelhante ao processo descrito no requisito R15. A única diferença é que o voto não é encriptado no dispositivo do eleitor mas sim no servidor eleitoral. Assim sendo, ao invés do utilizador enviar o boletim de voto encriptado, envia o mesmo em *plaintext*, através de um canal de comunicação seguro, para ser encriptado e registado.

7.11 R16 - Obter detalhe de um voto

Classificação: Must

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode obter o detalhe de votos em eleições criadas por si.

Funcionamento:

Todos os votos submetidos ao servidor eleitoral são armazenados encriptados no SGBD subjacente. De forma a garantir total transparência, e confiança no processo eleitoral no Helios, os votos encriptados são disponibilizados ao público sem qualquer restrição. Desde que a chave privada da eleição não seja conhecida ou obtida de alguma forma o conteúdo do voto permanece seguro.

Em eleições com apenas o *HeliosTrustee* a chave privada é detida pelo servidor eleitoral, mais concretamente no SGBD utilizado por este mesmo servidor. Desta forma, um eventual comprometimento do servidor eleitoral ou do SGBD poderá comprometer a eleição e revelar o sentido de voto dos votantes.

Diagrama de sequência

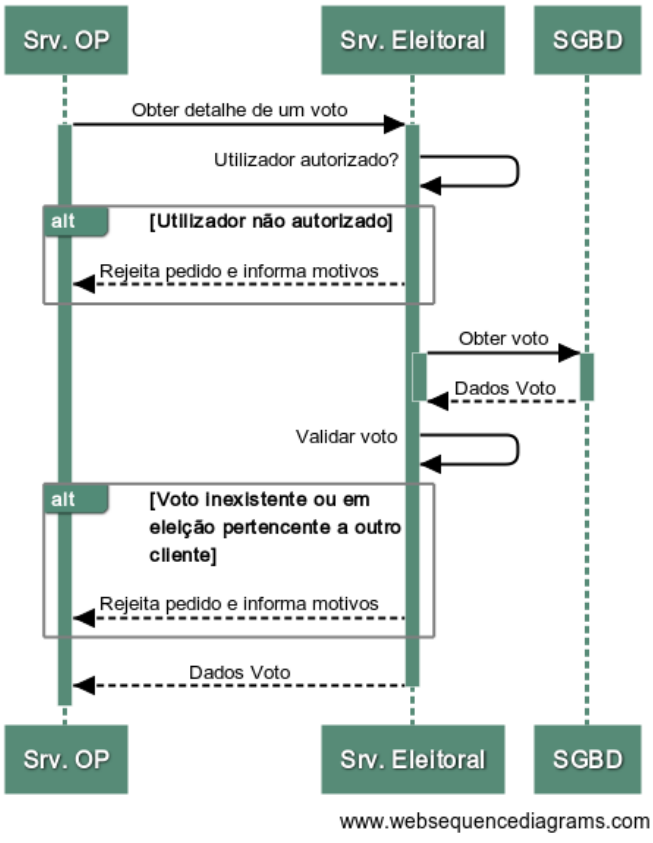

Figura 7.8: Diagrama sequência: Obter detalhe de um voto

7.12 R21 - Contabilizar os votos

Classificação: Must

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode contabilizar os votos de eleições criadas por si.

Funcionamento:

Através de um pedido POST os clientes da API podem solicitar ao servidor eleitoral a contabilização dos votos armazenados de uma eleição por estes criada. De seguida apresenta-se, resumidamente, o funcionamento deste processo.

1. Ao receber o pedido, a API consulta a base de dados de forma a obter os dados da eleição em causa. Após a obtenção dos dados da eleição, ir-se-á proceder à instanciação de um objeto da classe *Election* da biblioteca do Helios referida na secção 6.3.3. Para tal, ter-se-á que indicar a chave pública da eleição e as questões do boletim de voto.

2. De seguida, a API consulta a base de dados e obtém todos os eleitores que tenham exercido o direito de voto através da tabela Eleitores do SGBD. Nesta tabela constam não só as informações sobre o eleitor, como também o último voto, encriptado, submetido pelo eleitor ao servidor eleitoral.

3. Nesta fase, para cada um dos votantes, é instanciado um objeto da classe *Voter* da biblioteca do Helios referida na secção 6.3.3. Estas instâncias são armazenadas numa lista para serem processadas.

4. Depois de se ter a eleição e a lista de votantes (com o seu último voto) instanciados, ir-se-á proceder à contabilização dos votos encriptados. Como referido anteriormente, é possível proceder à contabilização dos votos encriptados graças à criptografia homomórfica. Esta contabilização é realizada através do método *compute_tally* da eleição. Este método recebe como único parâmetro os votantes e retorna o resultado da eleição encriptado.

5. Por fim, é registado o resultado encriptado da contabilização) no SGBD para ser, à posteriori, desencriptado. Note-se que em momento algum o servidor desencripta os votos.

Diagrama de sequência

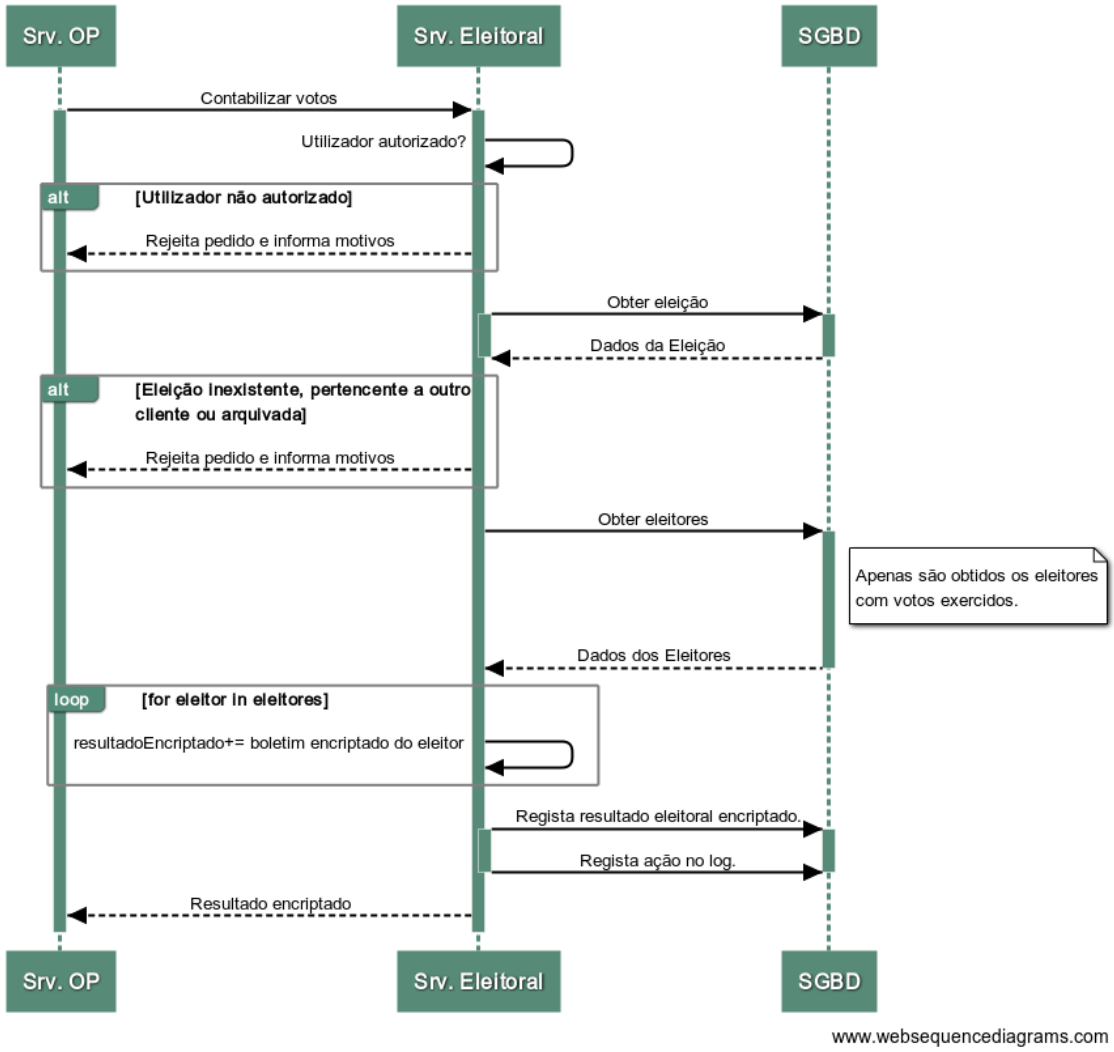

Figura 7.9: Diagrama sequência: Contabilizar os votos

7.13 R22 - Submeter os *decryption factors* e as *decryption proofs* dos *trustees*

Classificação: Must

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode submeter os *decryption factors* e as *decryption proofs* de *trustees* criados por si e, consequentemente, de eleições por si criadas.
- Os votos têm de ter sido previamente contabilizados.

Funcionamento:

Como referido anteriormente, todos os *trustees* da eleição são responsáveis por desencriptar o resultado da votação da eleição. Assim sendo, após a contabilização dos votos torna-se necessário que cada *trustee* proceda à desencriptação parcial do resultado da eleição. O *trustee* deverá, no seu terminal, proceder à desencriptação parcial do resultado da eleição. Neste processo o *trustee* gera e submete os *decryption factors* e as *decryption proofs* ao servidor através de um pedido POST. De seguida apresenta-se, resumidamente, o funcionamento deste processo.

1. Ao receber o pedido, a API consulta a base de dados de forma a obter os dados do *trustee* para o qual se submetem os *decryption factors* e as *decryption proofs*. Após a obtenção dos dados do *trustee*, nomeadamente da chave pública e do respetivo *hash*, ir-se-á proceder à instanciação de um objeto da classe *HeliosTrustee* da biblioteca do Helios, referida na secção 6.3.3, com esses mesmos dados e também com os *decryption factors* e as *decryption proofs* facultados no corpo do pedido.

2. Por fim, verificam-se, através do método *verify_ decryption_ proofs* do *HeliosTrustee*, as provas criptográficas fornecidas para a eleição e resultado eleitoral encriptado desta. Caso estas se apresentem válidas procede-se à atualização do registo dos *decryption factors* e das *decryption proofs* no SGBD. Por outro lado, caso as provas se apresentem inválidas, o pedido é rejeitado e é retornado o motivo pelo qual não pôde ser satisfeito.

Diagrama de sequência

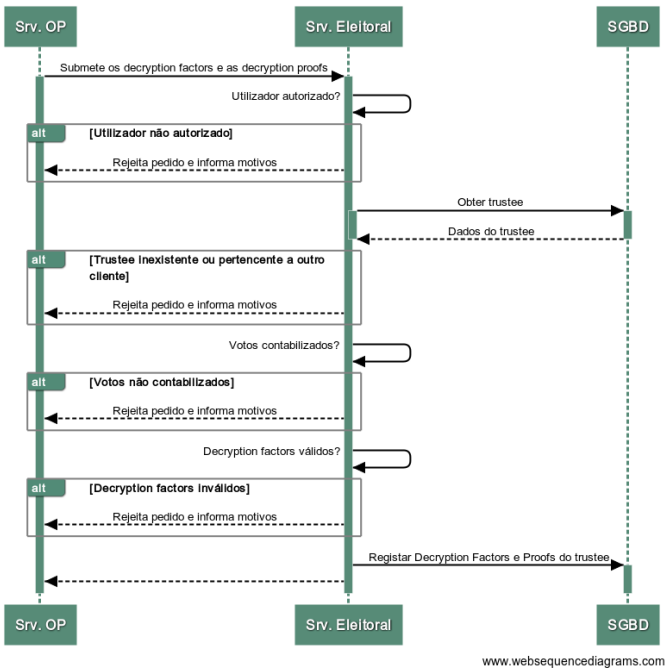

Figura 7.10: Diagrama sequência: Submeter os *decryption factors* e as *decryption proofs* dos *trustees*

7.14 R23 - Combinar os *decryption factors* dos *trustees*

Classificação: Must

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode combinar os *decryption factors* de *trustees* criados por si e, consequentemente, de eleições por si criadas.
- Os votos têm de ter sido previamente contabilizados.
- Todos os *trustees* têm de ter submetidos os seus *decryption factors*.

Funcionamento:

Após a submissão de todos os *decryption factors* por parte dos *trustees*, estão reunidas as condições para se poder combinar as desencriptações parciais e obter-se o resultado da eleição desencriptado. Para tal, um pedido através do método GET deverá ser feito ao servidor eleitoral. De seguida apresenta-se, resumidamente, o funcionamento do processo que irá permitir combinar os *decryption factors* e consequentemente obter o resultado desencriptado da eleição.

1. Ao receber o pedido a API consulta a base de dados de forma a obter a eleição, para a qual se pretende combinar os *decryption factors*, e também os *trustees* dessa eleição.
2. De seguida, torna-se necessário instanciar um objeto da classe *HeliosElection* indicando a chave pública da eleição obtida do SGBD.
3. O próximo passo é criar uma lista com objetos da classe *HeliosTrustee*. Para cada *trustee* um objeto será instanciado e inicializado com os *decryption factors*.
4. Tendo a lista de *Trustees* completa, tem que se associar ao objeto da classe *HeliosElection* instanciado e inicializado previamente. De seguida, ir-se-á criar e instanciar uma lista de listas que irá conter o resultado encriptado da eleição. Esta lista é por fim associada ao objeto da classe *HeliosElection* à semelhança do que fez com os *Trustees*.
5. Por fim, ir-se-á então proceder à combinação dos *decryption factors* propriamente dita. Como retorno, obter-se-á o resultado desencriptado da eleição. Para este processo, recorremos ao método *combine_ decryptions* do objeto da classe *HeliosElection*. O resultado produzido por este método é então armazenado no SGBD e retornado ao cliente.

Nota: Como referido anteriormente este processo só é possível se todos os *trustees* tiverem submetido os seus *decryption factors*. Para que os *trustees* possam submeter estes valores, eles precisam de conhecer a sua chave privada gerada aquando da sua criação. Se algum dos *trustees* não souber esta chave a desencriptação do resultado da eleição torna-se impossível e será necessário realizar nova eleição. Nesta fase importa reiterar a importância das

chaves privadas dos *trustees* e alertar para a necessidade de estas serem preservadas de forma segura até, pelo menos, ao final deste processo.

Diagrama de sequência

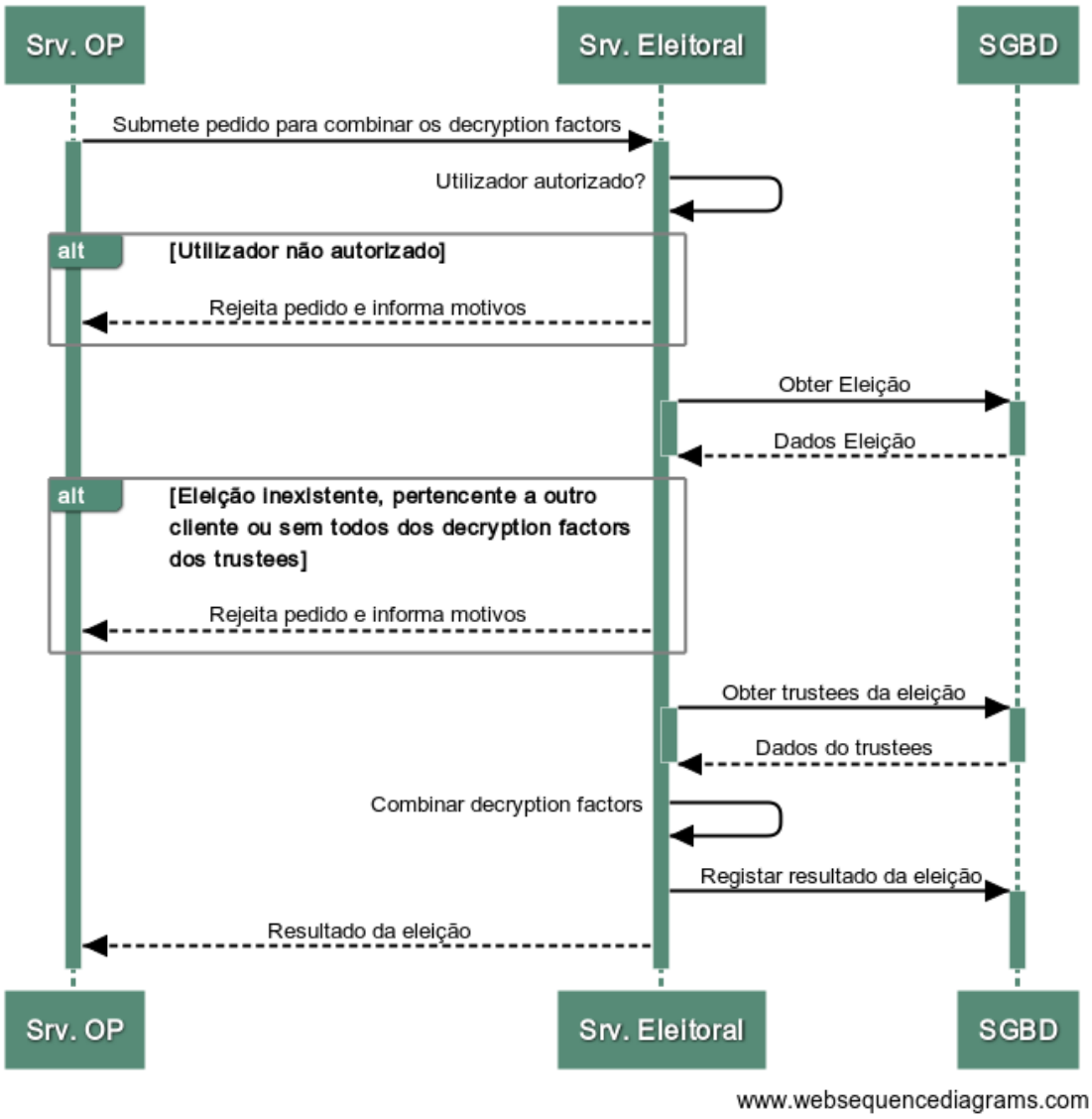

Figura 7.11: Diagrama sequência: Combinar os decryption factors dos *trustees*

7.15 R24 - Submeter voto auditado

Classificação: Could

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- Período de votação concluído

- Os votos têm de ter sido previamente contabilizados.

Funcionamento:

A grande parte do processo de auditoria de voto é realizada no terminal do votante. Os votos auditados devem, no entanto, ser registados no servidor eleitoral. Para tal, um pedido através do método POST, onde no corpo deste pedido consta o voto auditado, deverá ser feito ao servidor eleitoral. De seguida apresenta-se, resumidamente, o funcionamento do processo de registo do voto auditado.

1. Ao receber o pedido a API consulta a base de dados de forma a obter a eleição para a qual se deseja submeter o voto auditado.

2. De seguida, verificando-se que o período de votação terminou, ir-se-á então proceder à validação do voto auditado.

3. Estando o voto auditado válido procede-se então ao registo do mesmo no SGBD e é retornada uma mensagem de sucesso ao cliente.

Diagrama de sequência

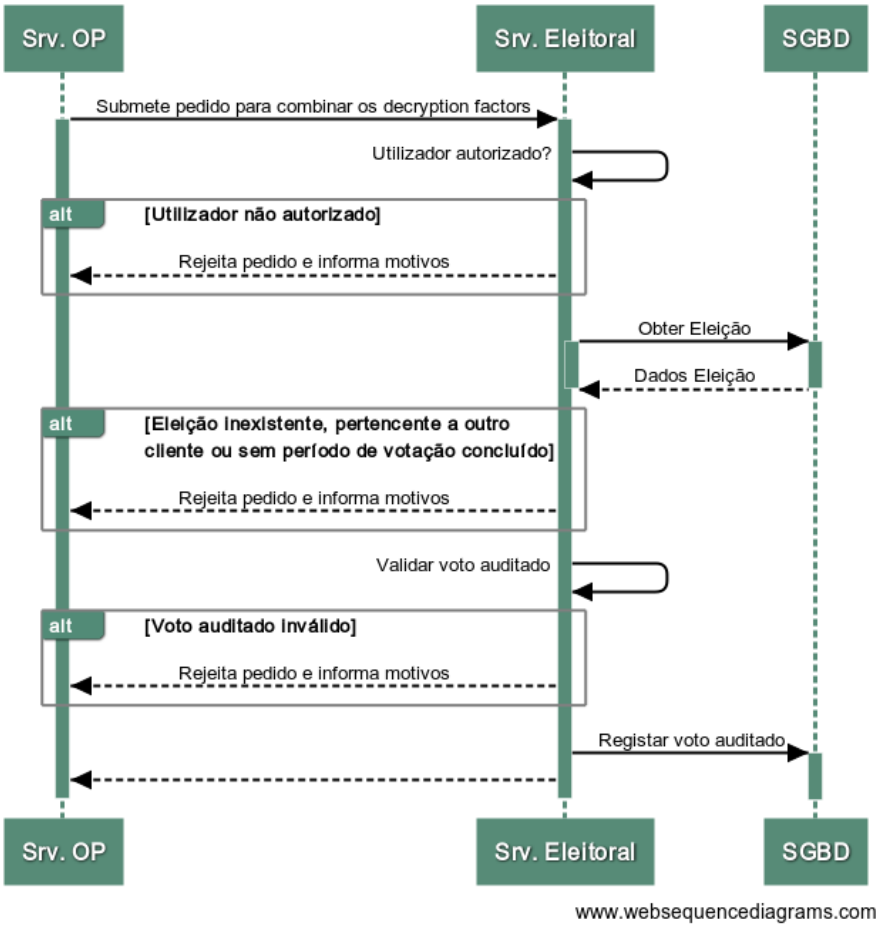

Figura 7.12: Diagrama sequência: Submissão de voto auditado

7.16 Conclusão

O presente capítulo iniciou-se pela definição, justificação e funcionamento do processo de autenticação dos clientes da *RESTful API* desenvolvida neste trabalho. Como se constatou optou-se pela utilização de chaves de *API* pois, para o cenário apresentado, considerou-se que seria uma forma de autenticação adequada.

Ao longo do capítulo pudemos ver a especificação e implementação dos diversos requisitos definidos. Na descrição de cada requisito apresentou-se a sua classificação, prioridade e restrições. Apresentou-se também o funcionamento geral desse requisito e o diagrama de sequência do mesmo.

Para concluir, importa referir que todos os requisitos, com a exceção dos classificados como *Won't*, foram implementados. Assim sendo, foram implementados 21 requisitos, de um total de 25, que permitem a gestão do processo de votação dos orçamentos participativos.

Capítulo 8

Validação da proposta

O presente capítulo pretende descrever os testes realizados afim de validar a solução proposta e implementada. Primeiramente, na secção 8.1, ir-se-ão descrever, de forma sucinta, alguns dos testes funcionais realizados afim de garantir o correto funcionamento da *RESTful API* desenvolvida. Por outro lado, na secção 8.2, apresentar-se-ão alguns dos testes efetuados de forma a avaliar a performance das várias funcionalidades da *RESTful API*. Por fim, apresentam-se algumas conclusões sobre os resultados obtidos.

8.1 Testes funcionais

Nesta subsecção pretendem-se apresentar alguns dos testes desenvolvidos. Os testes aqui descritos correspondem a uma ínfima parte do total de testes feitos com o objetivo de garantir o correto funcionamento e prevenir que eventuais alterações ao código possam comprometer o correto funcionamento do sistema.

Nota: Ao longo dos testes ir-se-ão apresentar todos os números criptográficos com apenas 40 caracteres. Todavia, na realidade estes possuem centenas de caracteres (600+).

Cada teste apresentado terá a seguinte estrutura: o caminho, o método Hyper Text Transfer Protocol (HTTP) [1], uma breve descrição do teste, uma breve descrição do resultado expectável e o corpo do pedido. O código dos testes aqui apresentados será disponibilizado no anexo J.

A realização dos testes descritos nesta subsecção recorre exclusivamente a módulos de testes fornecidos nativamente pela *Django Rest Framework (DRF)*. Em concreto, ir-se-á recorrer às classes *APITestCase* e a *APIClient*. Todas as funções de teste deverão iniciar com "*test*"pois esta é a convenção definida para que a ferramenta de testes possa saber quais os métodos a executar.

Como se referiu no capítulo anterior, todas as invocações à *RESTful API* devem estar devidamente autenticadas através de chaves de API. Para que os testes sejam possíveis, torna-se então necessário criar essa mesma chave para posterior utilização nos testes. Para tal, recorreu-se ao método *setUp* como se pode ver na figura 8.1.

[1] GET, POST, PUT, DELETE, HEAD, CONNECT, OPTIONS, TRACE e PATCH são, no momento da redação desta dissertação, os métodos disponibilizados pelo protocolo HTTP.

```python
class ElectionTestCases(APITestCase):

    def setUp(self) -> None:
        self.client = APIClient()
        obj, api_key = APIKey.objects.create_key(name='ApiKey')
        self.client.credentials(HTTP_AUTHORIZATION='Api-Key ' + api_key)
```

Figura 8.1: Criação da chave de API para execução dos testes

De seguida, apresentam-se dez dos testes realizados reiterando que estes são apenas uma ínfima parte da totalidade dos testes realizados.

8.1.1 Criação de eleição sem indicação de parâmetros

Caminho: /elections

Método HTTP: POST

Descrição:

A criação de eleições contém um conjunto de atributos que têm que obrigatoriamente ser facultados.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é rejeitado.
- Na resposta é indicado o código HTTP correto.
- O corpo da resposta indica, para cada atributo, o conjunto de regras que estão a ser violadas (no caso em concreto pela obrigatoriedade do preenchimento).

Dado o funcionamento das validações dos modelos de dados na DRF, este teste não precisa de validar individualmente cada um dos atributos obrigatórios uma vez que esta retorna, para todos os atributos que violem as regras estabelecidas, as regras que estão a ser violadas.

Corpo de pedido

O corpo do pedido é constituído por um objeto, em notação JSON, sem qualquer atributo.

```
{}
```

Resultado expectável:

Pedido rejeitado com o código HTTP 400 e corpo da resposta em formato JSON com a indicação de que os seguintes atributos são obrigatórios: *short_name*, *name* e *description*.

Exemplo de corpo expectável da resposta:

```
{
    "short_name": ["This field is required."],
    "name": ["This field is required."],
    "description": ["This field is required."]
```

```
}
```

8.1.2 Criação de eleição

Caminho: /elections

Método HTTP: POST

Descrição:

O processo de criação de uma eleição pressupõe, desde que os dados facultados no corpo do pedido estejam de acordo com as regras definidas, a criação e armazenamento do *HeliosTrustee* e respetivas chaves criptográficas.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- O *HeliosTrustee* é criado e armazenado no SGBD.
- Os valores por omissão para a eleição e *HeliosTrustee* estão definidos corretamente.
- Na resposta é indicado o código HTTP correto.
- O corpo da resposta é constituído pelos atributos públicos da eleição.
- Se os atributos no corpo da resposta têm os valores expectáveis.

Corpo de pedido

O corpo do pedido é constituído pelos atributos obrigatórios de uma eleição (*short_ name*, *name* e *description*)

```
{
    "short_name": "OP2020",
    "name": "Orcamento Participativo 2020",
    "description": "Neque porro quisquam est qui dolorem ipsum
        quia dolor sit amet, consectetur, adipisci velit..."
}
```

Resultado expectável:

Pedido processado, resposta com o código HTTP 201 e corpo da resposta em formato JSON com os atributos públicos da eleição.

Exemplo de corpo expectável da resposta:

```
{
"uuid": "f0c74d8a-5768-4139-9f8f-82b3766efa32",
"short_name": "OP2020",
"name": "Orcamento Participativo 2020",
"description": "Neque porro quisquam est qui dolorem ipsum
   quia dolor sit amet, consectetur, adipisci velit...",
```

```
"election_type": "election",
"createdAt": "2020-08-15T02:49.525746",
"lastModifiedAt": "2020-08-15T02:49:49.525746",
"archived_at": null,
"frozen_at": null,
"public_key": null,
"questions": [],
"use_voter_aliases": false,
"use_advanced_audit_features": true,
"randomize_answer_order": false,
"openreg": false,
"voters_hash": null,
"voting_extended_until": null,
"registration_starts_at": null,
"voting_starts_at": null,
"voting_ends_at": null,
"voting_started_at": null,
"voting_ended_at": null,
"tallying_started_at": null,
"tallying_finished_at": null,
"tallies_combined_at": null,
"result_released_at": null,
"encrypted_tally": null,
"result": null,
"result_proof": null
}
```


8.1.3 Adicionar um trustee

Caminho: /elections/:id/trustees

Método HTTP: POST

Como referido, aquando a criação de uma eleição é criado um *trustee*, denominado de *HeliosTrustee*, por omissão. Todavia, em processos eleitorais que requeiram um maior nível de segurança, é possível adicionar mais *trustees* a uma eleição enquanto esta não tiver sido congelada.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- Um *trustee* pode ser adicionado a uma eleição desde que esta não esteja congelada.
- Na resposta é indicado o código HTTP correto.
- O corpo da resposta é constituído pelos atributos públicos do *trustee*.
- Se os atributos no corpo da resposta têm os valores expectáveis.

Corpo de pedido

O corpo do pedido é constituído pelos atributos obrigatórios para o registo de um novo *Trustee* numa eleição (*name*, *public_key* e a prova de conhecimento da chave privada *pok*).

```
{
    "name": "Leandro"
    "public_key":{
        "p": "16328632084933010002384055033805457329 60",
        "q": "35596086693371311393755083704587789171 56",
        "g": "14887492224963187634282421537186040801 30",
        "y": "63132922193684541072283520638276992153 34"
    },
    "pok": {
        "response": "48977110093550721921610548274695736566 40"
        ,
        "challenge": "11965689960663114510730930134288981521 57
        ",
        "commitment": "15539963118536927432918372869151845801 75"
    },
}
```

Resultado expectável:

Pedido processado, resposta com o código HTTP 201 e corpo da resposta em formato JSON com os atributos públicos do *Trustee*.

Exemplo de corpo expectável da resposta:

```
{
  "uuid": "9031a58a-7864-48d5-8eff-ddf6c778d25b",
  "name": "Leandro",
  "public_key": {
    "y": "16328632084933010002384055033805457329 60",
    "p": "35596086693371311393755083704587789171 56",
    "g": "14887492224963187634282421537186040801 30",
    "q": "63132922193684541072283520638276992153 34"
  },
  "pok": {
    "response": "48977110093550721921610548274695736566 40",
    "challenge": "11965689960663114510730930134288981521 57",
    "commitment": "15539963118536927432918372869151845801 75"
  },
  "public_key_hash": "f0c74d8a-5768-4139-9f8f-82b3766efa32",
  "election": "943c26f6-d001-4d58-84da-65c95ce2ea17"
}
```

8.1.4 Adição de um eleitor à eleição

Caminho: /voters

Método HTTP: POST

Descrição

Num processo eleitoral torna-se necessário o registo de eleitores. Dependendo da configuração estes podem ser adicionados até ao congelamento da eleição ou, se a eleição for aberta, os eleitores podem ser adicionados até ao momento em que se procede à contabilização dos votos. Neste teste, ir-se-á verificar se a adição de um eleitor a uma eleição, numa eleição restrita e não congelada, é possível.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- Um eleitor pode ser adicionado a uma eleição desde que esta não esteja congelada.
- Na resposta é indicado o código HTTP correto.
- O corpo da resposta é constituído pelos atributos públicos de um eleitor.
- Se os atributos no corpo da resposta têm os valores expectáveis.

Corpo de pedido

O corpo do pedido é constituído por um objeto, em notação JSON, onde constam o identificador da eleição onde se pretende registar o eleitor (*Universally unique identifier (UUID)*), o identificador do eleitor no orçamento participativo (*voter_id*) e o nome do eleitor (*name*).

```
{
    "election": "f0c74d8a-5768-4139-9f8f-82b3766efa32",
    "voter_id": "V3",
    "name": "Leandro"
}
```

Resultado expectável:

Pedido processado, resposta com o código HTTP 201 e corpo da resposta em formato JSON com os atributos públicos do eleitor.

Exemplo de corpo expectável da resposta:

```
{
  "uuid": "1a3409f4-35b0-4e5d-9faa-f870738cbfe7",
  "voter_id": "V3",
  "name": "Leandro",
  "alias": null,
  "vote": null,
  "vote_hash": null,
  "cast_at": null,
  "election": "f0c74d8a-5768-4139-9f8f-82b3766efa32",
}
```

8.1.5 Adição de um eleitor à eleição com o mesmo identificador

Caminho: /voters

Método HTTP: POST

Descrição

Dentro de uma eleição o *voter_id*, identificador do eleitor no orçamento participativo, tem que ser único. Esta foi uma regra que se definiu aquando o desenvolvimento da *RESTfull API*. Neste sentido, neste teste ir-se-á primeiramente adicionar um eleitor e após a adição deste ir-se-á executar um outro pedido mudando apenas o atributo *name*. Tendo em conta que tanto o atributo *election* como o *voter_id* permanecem iguais, estamos perante a violação da referida regra.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é rejeitado.
- Na resposta é indicado o código HTTP correto.

Corpo de pedido

O corpo do pedido é constituído por um objeto, em notação JSON, onde constam o identificador da eleição onde se pretende registar o eleitor (*UUID*), o identificador do eleitor no orçamento participativo (*voter_id*) e o nome do eleitor (*name*).

```
{
    "election": "f0c74d8a-5768-4139-9f8f-82b3766efa32",
    "voter_id": "V3",
    "name": "Rodrigues"
}
```

Resultado expectável:

Pedido rejeitado, resposta com o código HTTP 400.

Exemplo de corpo expectável da resposta:

```
{
    "non_field_errors": "The fields election, voter_id must
       make a unique set."
}
```

8.1.6 Congelamento de uma eleição

Caminho: /elections/:id/freeze

Método HTTP: PUT

Descrição:

O processo de "congelar"uma eleição é bastante importante na proposta do Helios. É neste processo que se dá a geração da chave pública da eleição, através da combinação das chaves públicas dos *trustees* registados para essa eleição, que será utilizada para encriptar o boletim de voto do eleitor e garantir o anonimato do voto.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- A eleição é congelada.
- As chaves públicas dos seus *trustees* são combinadas dando origem à chave pública da eleição.
- Na resposta é indicado o código HTTP correto.

Corpo de pedido

O corpo do pedido é constituído por um objeto, em notação JSON, sem qualquer atributo.

```
{}
```

Resultado expectável:

Pedido processado, resposta com o código HTTP 200. No corpo da resposta deverão constar todos os dados públicos da eleição criada nos quais se deve incluir, pela primeira vez, a chave pública da eleição.

Exemplo de corpo expectável da resposta:


```
{
"uuid": "f0c74d8a-5768-4139-9f8f-82b3766efa32",
"short_name": "OP2020",
"name": "Orcamento Participativo 2020",
"description": "Neque porro quisquam est qui dolorem ipsum
   quia dolor sit amet, consectetur, adipisci velit...",
"election_type": "election",
"createdAt": "2020-08-15T02:49.525746",
"lastModifiedAt": "2020-08-15T03:15:32.47372",
"archived_at": null,
"frozen_at": "2020-08-15T03:15:32.47372",
 "public_key": {
    "y": "16328632084734010002384055033805457329 60",
    "p": "35596083699371311393755083704587789171 56",
    "g": "14887492284963487634282421537186040801 30",
    "q": "13135922793684541072283520638276992153 34"
  },
"questions": [{"question": "Por favor selecione um projeto..."
   , "answer_urls": ["https://site.pt/1", "https://site.pt/2"]
   , "answers": ["Projeto A", "Projeto B"], "choice_type": "
   approval", "max": 1, "min": 1, "result_type": "absolute", "
   short_name": "", "tally_type": "homomorphic"}]
```

```
"use_voter_aliases": false,
"use_advanced_audit_features": true,
"randomize_answer_order": false,
"openreg": false,
"voters_hash": null,
"voting_extended_until": null,
"registration_starts_at": null,
"voting_starts_at": "2020-08-20T23:00:00"
"voting_ends_at": "2020-08-30T22:59:00"
"voting_started_at": null,
"voting_ended_at": null,
"tallying_started_at": null,
"tallying_finished_at": null,
"tallies_combined_at": null,
"result_released_at": null,
"encrypted_tally": null,
"result": null,
"result_proof": null
}
```

8.1.7 Adição de um voto encriptado

Caminho: /elections/:id/cast_vote

Método HTTP: POST

Descrição

O exercício do voto está intrinsecamente relacionado com processos eleitorais. De facto, este exercício é dos atos mais importantes nos processos eleitorais e também na democracia.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- O voto é validado, processado e armazenado.
- É gerado um *hash* criptográfico do voto.
- Na resposta é indicado o código HTTP correto.

Corpo de pedido

O corpo do pedido é constituído pelo *encrypted_vote* (voto encriptado) e pelo UUID do eleitor. Tendo em conta que o voto encriptado tem uma enorme dimensão, o que impossibilita a sua apresentação, optou-se por apresentar apenas a estrutura, em formato JSON.

```
{
    "encrypted_vote": {
    "answers": [
        {
```

```
            "choices": [
                {
                    "alpha": "29811835580141775763514107250284
                       19674963",
                    "beta": "36681617620922435584200673511588
                       4474396"
                },
                ...
            ],
            "individual_proofs": [
                [
                    {
                        "commitment": {
                            "A": "90210796615515549547922425412
                               10664280056",
                            "B": "45513553853169126329308511078
                               81210473799"
                        },
                        "challenge": "72966200464603682331925618
                           20659268017825430543792144429376789655561017578919",
                        "response": "48159134082256723219342052
                           73913358598029639880150659065335192
                           0141412518813404"
                    },
                   ...
                ],
               ...
            ],
            "overall_proof": [
                {
                    "commitment": {
                        "A": "20169003273964199673880531551158
                           23754685",
                        "B": "78247600402721844760619509112158
                           96448868"
                    },
                    "challenge": "51083280484456296985053092183949648725833779953244158238309239115512883141473",
                    "response": "50008780776147780721689521361729033411298670740278348031444265090792174355207"
                },
              ...
            ],
            "randomness": [
                "47503712377201154160509345149634754046457222746182731793470959197618733635966",
                "42648651797616251863569678299866863077627445696875924660707612973223728343401",
```

```
                "426111227966559896599667678458741051964722693
                   32482196293023745987495904963924"
            ]
        }
    ],
    "election_hash": null,
    "election_uuid": "f0c74d8a-5768-4139-9f8f-82b3766efa32"
},
"voter_uuid":"1f3c9249-1281-4cfc-ade3-77b151d1be02"
}
```

Resultado expectável:

Pedido processado com o código HTTP 201. Voto armazenado na tabela de votos e também na tabela de eleitores associado ao eleitor que realizou o voto.

Exemplo de corpo expectável da resposta:

```
{
    "vote_hash": "YTLrAm3hUU0yxfKHJFcRK9+eQFuUAoTu7ltGQNnNVZA"
}
```

8.1.8 Adição de um voto encriptado inválido

Caminho: /elections/:id/cast_vote

Método HTTP: POST

Descrição:

Neste teste pretende-se verificar se o servidor eleitoral é capaz de reconhecer e rejeitar um voto encriptado mal constituído. Este é apenas um dos vários testes efetuados ao processo de adição de voto. Por simplicidade, apresenta-se apenas este teste. Todavia, testes com um votante inexistente, com diversos atributos do voto encriptado inválido, tentativa de voto fora do período eleitoral, entre outros, foram realizados no sentido de procurar garantir o correto funcionamento deste processo.

Com este teste, espera-se garantir o seguinte:

- O pedido é rejeitado.
- É gerado um *hash* criptográfico do voto.
- Na resposta é indicado o código HTTP correto.

Corpo de pedido

O corpo do pedido é em todo semelhante ao do teste anterior com a excepção das *individual_proofs*. Nestas alterar-se-á apenas um digito na primeira *individual_proof*.

Resultado expectável:

Pedido rejeitado com o código HTTP 400. Como uma das *individual_proofs* foi manipulada, espera-se que o servidor rejeite o pedido.

Exemplo de corpo expectável da resposta:

```
{
    "encrypted_vote": ["The provided encrypted vote is not
       valid"]
}
```

8.1.9 Contabilização dos votos

Caminho: /elections/:id/compute_tally

Método HTTP: POST

Descrição:

O processo de contabilização de votos é um dos mais importantes. Durante o período de votação os eleitores, de acordo com as regras, foram votando nas suas preferências. Após este período, chega a aguardada contabilização dos votos para posterior publicação.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- Os votos são contabilizados e o o seu resultado encriptado é registado na eleição.
- Uma vez que na eleição criada o *HeliosTrustee* é o único *Trustee*, pretende-se garantir que a eleição fica pronta para ser desencriptada.
- Na resposta é indicado o código HTTP correto.

Recorde-se que os votos armazenados no servidor estão encriptados com a chave pública da eleição. A contabilização dos votos é possível, uma vez que se recorre a criptografia homomórfica, sem ter que se desencriptar os mesmos.

Note-se que, após a contabilização dos votos, é também realizada a desencriptação parcial do resultado encriptado, para o *HeliosTrustee*. Se mais *trustees* estiverem associados à eleição, terão que processar e submeter as suas desencriptações parciais.

Corpo de pedido

O corpo do pedido é constituído por um objeto, em notação JSON, sem qualquer atributo.

```
{}
```

Resultado expectável:

Pedido processado com código HTTP 200. Resultado encriptado da eleição armazenado na tabela *elections* do SGBD. Adicionalmente, espera-se também que a desencriptação parcial para o *HeliosTrustee* seja determinada e armazenada no registo deste *Trustee*.

Exemplo de corpo expectável da resposta:

```
{
  "ready_for_decryption_combination": true,
  "encrypted_tally": {
    "tally": [
      [
        {
          "alpha": "13746899818529535974424792769610972439 7",
          "beta": "11000801277886734472765918362516412429 99"
        },
        {
          "alpha": "27239785216894889303069055087185280767 91",
          "beta": "36636897296554716624540595863611185017 54"
        }
      ]
    ],
    "num_tallied": 4
  }
}
```


8.1.10 Combinação dos resultados parciais e desencriptação dos resultados

Caminho: /trustees/:id/combine_decryption

Método HTTP: POST

Descrição:

O processo de combinação dos resultados parciais é dos últimos processos mas um dos mais importantes, pois é através dele que se obtêm os resultados desencriptados da eleição. Assim que todos os *trustees* tenham os seus resultados parciais, estão reunidas as condições para se proceder à combinação destes mesmos resultados de forma a que se obtenha o resultado, desencriptado, da eleição.

Como tal, com este teste, espera-se garantir o seguinte:

- O pedido é aceite.
- O resultado eleitoral é desencriptado e apresenta valor correto.
- Na resposta é indicado o código HTTP correto.

Corpo de pedido

O corpo do pedido é constituído por um objeto, em notação JSON, sem qualquer atributo.

```
{}
```

Resultado expectável:

Pedido processado com código HTTP 200. Corpo da resposta com o resultado desencriptado da eleição.

Exemplo de corpo expectável da resposta:

```
{
  "result": [[0, 2]]
}
```

8.2 Testes não funcionais

Nesta secção ir-se-ão apresentar alguns dos testes executados. Para a sua realização ir-se-á recorrer ao serviço Loader.io. A escolha deste serviço tem que ver essencialmente com o facto de ser um serviço que tem vindo a ser utilizado na empresa, que permite 3 tipos de testes diferentes (clientes por testes, clientes por segundo e um número de clientes constante) e que, na versão gratuita, permite testes com até dez mil clientes.

O principal objetivo dos testes que aqui irão ser apresentados é perceber os tempos médios de resposta que, para a configuração apresentada e para um dado número de pedidos/utilizadores num determinado espaço de tempo, se consegue obter. Desta forma, poder-se-á verificar e saber com uma maior precisão os recursos que necessitam de ser alocados para um processamento mais expedito e com menos falhas para um previsível número de pedidos.

Criação e configuração de um teste

O serviço Loader.io disponibiliza uma plataforma web que, de forma bastante simples, intuitiva e parametrizável, permite a criação e configuração de três tipos de testes. São eles:

- ***Clients per test*** - Neste tipo de teste um número N de clientes é distribuído, o mais uniformemente possível, durante a duração do teste. A título de exemplo, se definirmos o N=10000 e a duração do teste em 20 segundos 500 clientes por segundo ir-se-ão conectar o que se traduzirá em 500 pedidos por segundo.
- ***Clients per second*** - Este é um tipo de testes bastante parecido com o anterior. A diferença reside no facto de neste, ao contrário do anterior, se definir o número de clientes por segundo ao invés do total de clientes para a duração total do teste. Um teste com 500 clientes por segundo com uma duração de 20 segundos é igual a ter o teste do exemplo anterior.
- ***Maintain client load*** - Este tipo de teste permite especificar um valor inicial e final de clientes. A título de exemplo, podemos definir um teste de 10 a 100 clientes. Este irá iniciar com 10 clientes e vai aumentando o número à medida que o teste vai decorrendo. A principal diferença neste teste, em relação aos restantes, é que cada cliente irá continuamente emitir pedidos até ao final do teste ao invés de parar após um pedido.

A criação de um teste exige a **definição** do **nome**, **tipo de teste** a realizar, **número de clientes**, **duração** do teste e, por fim, a **definição dos pedido(s)** das rotas a testar. Para cada rota é necessário facultar o método HTTP, o protocolo, o *host* e a rota a testar. Opcionalmente, podem ser definidos os *headers* do pedido, parâmetros e corpo do pedido (quando aplicável) e variáveis de resposta se existir a pretenção de verificar se um

determinado atributo da resposta tem um determinado valor. Nos testes que realizaremos ir-se-á recorrer essencialmente à definição do *header "Authorization"* onde se definirá a chave de *API*, e também à definição do corpo do pedido que será, na generalidade, definido através de um ficheiro com estrutura definida pelo serviço Loader.io. Note-se que através do ficheiro podemos passar um qualquer número de parâmetros por cada pedido que é feito. A interface para criação de um teste através do Loader.io pode ser consultada no anexo K.

Ambiente de testes

É expectável que a *RESTfull API* desenvolvida corra dentro de um *cluster* [2] de servidores geridos pelo Kubernetes [3]. O *Kubernets cluster* será constituído por máquinas virtuais dedicadas com as seguintes especificações: Processador intel com 4 Virtual CPU (vCPUs), 8GB RAM e 50GB de disco rígido. Os testes que se irão apresentar de seguida serão realizados numa instância, com o *docker engine* instalado, com as referidas especificações. Nessa instância correrão quatro *containers docker* com a *RESTfull API* desenvolvida. A base de dados necessária, PostgreSQL versão 12, irá correr numa instância à parte mas dentro da mesma Virtual Private Cloud (VPC).

8.2.1 Criação de eleições

Uma das funcionalidades basilares de um servidor eleitoral é a criação de eleições. Neste sentido, pretende-se avaliar o desempenho do servidor perante a submissão de N eleições ao longo de 30 segundos. Os resultados dos testes efetuados, com diferentes valores para o N, encontram-se na tabela 8.1.

Para levar a cabo a execução deste teste foi necessário, recorrendo à criação de um pequeno *script*, gerar um conjunto de atributos para injetar em cada um dos pedidos. Através desse *script* procedeu-se à geração de 300 eleições que serviram para a realização dos testes aqui apresentados. Depois, esse conjunto de dados foi disponibilizado publicamente, através do serviço S3, e o *url* de acesso foi indicado numa caixa de texto para o efeito presente na interface do Loader.io.

Como se poderá constatar através da referida tabela, à medida que se aumenta o número de eleições a criar de acordo no intervalo de 30 segundos, verifica-se um aumento dos tempos médios de resposta. Tal aumento é expectável pois o servidor tem de processar um maior número de pedidos que, no caso em concreto, envolve inclusive a geração do *HeliosTrustee* e de um par de chaves criptográficas. No geral os tempos obtidos são muito positivos. Apenas a partir dos 250 pedidos em 30 segundos se verificam tempos médios superiores a 3s. Todavia, este é um valor bastante bom e que é perfeitamente aceitável. Note-se que o processo de criação de eleições é feito, por norma, apenas uma vez por ano e por cliente.

[2] Um *cluster* é um conjunto de servidores interligados que funcionam como um grande sistema. Estes são essenciais para se conseguir escalabilidade horizontal.

[3] Kubernetes é uma plataforma de código aberto para a automatização e orquestração de *containers* desenvolvida pela Google.

	Tempos de resposta		
Eleições (N)	**Min**	**Max**	**Med**
30	346	728	365
60	343	745	406
90	346	887	481
150	481	1033	651
180	369	1109	741
210	580	1363	868
240	894	2482	1591
250	1068	3748	2575
260	1163	5792	3461
270	1252	6495	4111
300	1128	8951	5511

Tabela 8.1: Tempos de resposta na criação de N eleições em 30 segundos

8.2.2 Adição de eleitores a eleições

As eleições só fazem sentido porque existe um conjunto de pessoas, denominadas eleitores, que pretendem fazer parte do processo eleitoral e exercer o direito de voto. Assim sendo, pretende-se avaliar o desempenho do servidor perante o registo de N eleitores ao longo de 30 segundos. Os resultados dos testes efetuados, com diferentes valores para o N encontram-se na tabela 8.2.

Assim, como no teste anterior, foi necessário proceder-se à criação de um *script* para a geração do nome e do identificador do votante no orçamento participativo (*voter_id*). Para a geração das centenas de identificadores, que tinham de ser obrigatoriamente únicos, foi feita recorrendo à geração de um identificador único universal[4] através do algoritmo uuid4.

Tal como expectável, à medida que o número de eleitores a registar no intervalo de 30 segundos aumentou também os tempos médios aumentaram. Através deste teste foi possível verificar que o desempenho no registo de eleitores é suficiente para a cadência de registo que se tem verificado nas aplicações do orçamento participativo. O desempenho verificado é inferior ao verificado aquando a criação de eleições. Após análise ao código e aos tempos de execução das várias instruções, foi possível verificar que as transações e o bloqueio (*lock*) imposto à eleição, para evitar *race conditions* na base de dados, leva a um desempenho inferior ao esperado mas, como referido, suficiente para a cadência expectável (de acordo com dados históricos).

[4]Um identificador único universal (do inglês *universally unique identifier* - UUID) é um número de 128 bits utilizado para identificar informações em sistemas computacionais. Este é geralmente codificado através do sistema de numeração hexadecimal e o seu principal objetivo é permitir que sistemas distribuídos possam identificar univocamente informações sem coordenação central.

	Tempos de resposta		
Eleitores (N)	**Min**	**Max**	**Med**
30	338	635	368
60	340	880	464
90	336	965	487
150	456	1262	734
180	840	2622	1677
210	858	6285	4402
240	992	9055	5656
250	960	9344	5714
260	960	10487	6291
270	761	10666	6592
300	884	13318	7434

Tabela 8.2: Tempos de resposta no registo de N eleitores em 30 segundos

8.2.3 Congelamento de eleições

Antes de uma eleição poder ser votada esta deve ser devidamente configurada e, por fim, congelada de forma a que não seja possível a realização de quaisquer alterações e também para que a geração da chave pública da eleição aconteça. A votação só poderá acontecer se a eleição estiver congelada e, assim sendo, esta é uma funcionalidade muito importante que é realizada uma vez por cada eleição. Como tal, pretende-se avaliar o desempenho do servidor perante o pedido de congelamento de N eleições ao longo de 30 segundos. Os resultados dos testes efetuados, com diferentes valores para o N encontram-se nas tabelas 8.3 e 8.4.

Para testar o desempenho do processo procedeu-se ao registo faseado de eleições com 1000, 10K, 20K, 30K, 40K e 50K eleitores. O objetivo é verificar os tempos médios de acordo com o número de eleitores e número de pedidos de congelamento num período de 30s. Os resultados verificados são realmente muito positivos. Analisando a referida tabela verifica-se que o aumento de número de eleitores não se traduz num direto aumento substancial dos tempos médios de resposta. De facto, é o aumento do número de pedidos de congelamento de eleições no intervalo de 30s que faz, justificávelmente, aumentar os tempos de resposta. Dizemos justificávelmente porque neste processo existe a geração de um *hash* criptográfico dos dados dos eleitores que exige alguns recursos computacionais e, que com o aumento dos pedidos, sobrecarrega o servidor. Note-se que é no congelamento de 300 eleições que se verifica um maior incremento nos tempos médios de resposta. Através do gráfico apresentado na figura 8.2 é possível verificar-se exatamente a evolução dos tempos médios.

	1000 eleitores			10000 eleitores			20000 eleitores		
	Tempos de resposta			Tempos de resposta			Tempos de resposta		
Eleições (N)	**Min**	**Max**	**Med**	**Min**	**Max**	**Med**	**Min**	**Max**	**Med**
15	217	803	262	225	615	260	225	518	258
30	219	524	234	226	506	254	222	574	238
60	221	509	271	231	621	275	232	578	284
90	219	662	298	221	615	313	221	649	304
150	313	686	393	311	768	407	296	646	381
180	277	707	385	344	725	432	346	776	443
250	221	863	487	229	907	495	224	840	450
300	368	964	638	355	972	657	374	995	645

Tabela 8.3: Tempos de resposta no congelamento de eleições com 1000, 10K e 20K eleitores

	30000 eleitores			40000 eleitores			50000 eleitores		
	Tempos de resposta			Tempos de resposta			Tempos de resposta		
Eleições (N)	**Min**	**Max**	**Med**	**Min**	**Max**	**Med**	**Min**	**Max**	**Med**
15	232	538	257	238	568	268	270	674	302
30	227	490	246	233	507	259	262	565	279
60	227	663	273	232	627	281	270	545	315
90	225	579	301	231	588	316	271	707	367
150	340	727	407	342	730	419	367	830	474
180	353	733	448	359	794	454	401	876	545
250	232	873	495	229	850	482	273	1014	629
300	354	1007	652	376	1037	680	495	1471	810

Tabela 8.4: Tempos de resposta no congelamento de eleições de 30K, 40K e 50K eleitores

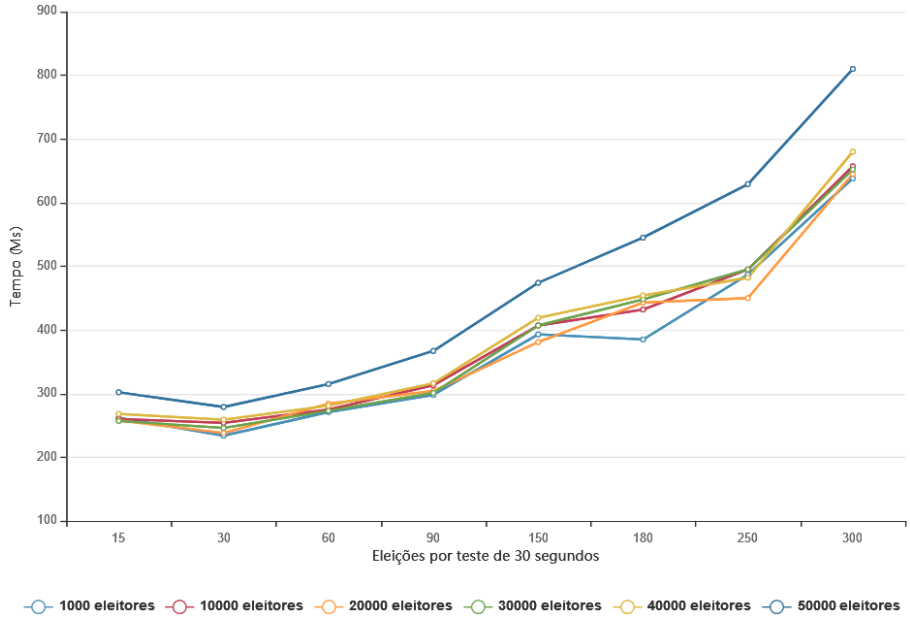

Figura 8.2: Evolução dos tempos resposta no congelamento de eleições

8.2.4 Adição de votos encriptados

O exercício do voto é fundamental em qualquer processo eleitoral. Assim sendo, pretende-se avaliar o desempenho do servidor perante o registo de N votos encriptados ao longo de 30 segundos. Os resultados dos testes poderão encontram-se na tabela 8.5.

Antes de se proceder à realização do teste teve que se proceder à geração e registo de eleitores nas eleições que irão fazer parte do teste. Este procedimento foi feito com a ajuda de um pequeno *script* que gerava e registava, diretamente na base de dados, o número de eleitores facultado. Tal como nos testes anteriores, foi também necessário proceder-se à geração de um conjunto de votos encriptados através de um *script* criado para o efeito. Esses dados foram então registados num ficheiro que foi disponibilizado ao Loader.io para que este, em cada pedido, registasse um voto para um eleitor diferente e com os parâmetros do corpo da mensagem distintos.

É objetivo deste teste determinar o desempenho do servidor quando confrontado com um número N de pedidos de registo de votos encriptados. Assim sendo, procedeu-se ao registo de 30, 60, 90, 150, 180, 210, 250 e 300 votos ao longo de 30 segundos. Como se poderá constatar, os resultados obtidos são os mais negativos até ao momento. Tal era expectável pois no processo de registo de um voto encriptado existe a necessidade de se garantir que o voto, mesmo encriptado, cumpre as regras definidas para a respetiva eleição. Dado que o voto está encriptado essas validações têm que ser feitas recorrendo a algoritmos criptográficos que, expectávelmente, consomem consideráveis recursos computacionais (em especial CPU).

Analisando os dados da tabela facilmente se percebe que até 180 votos (consultar gráfico presente na figura 8.3), num período de 30 segundos, o servidor apresenta tempos de resposta abaixo de 2s, que é um valor bastante positivo. Todavia, à medida que se aumenta o número de votos a registar no mesmo período verifica-se um considerável aumento especilmente nos tempos máximos e médios. Tal comportamento é justificado pela saturação do CPU que fica com uma utilização praticamente nos 100%. A partir dos 300 votos em 300 segundos verificou-se ainda que alguns dos pedidos podem ser descartados por não haver capacidade de processamento. Os valores obtidos parecem ser mais do que suficientes em especial porque existirá um *cluster* de servidores o que permitirá elevar a capacidade e minimizar os tempos de resposta.

	Tempos de resposta		
Votos	**Min**	**Max**	**Med**
30	617	905	631
60	609	1102	692
90	627	1103	796
150	906	1724	1055
180	940	2192	1485
210	1025	5415	4562
250	1380	11795	7202
300	1809	14895	8967

Tabela 8.5: Tempos de resposta na adição de votos encriptados numa eleição

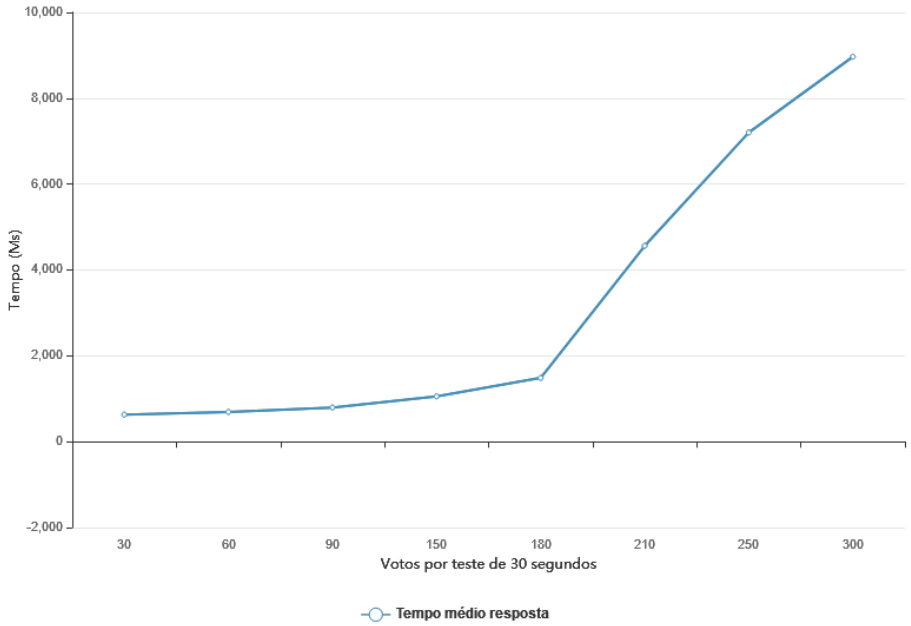

Figura 8.3: Evolução dos tempos de resposta no registo de voto

8.2.5 Contabilização de votos encriptados

A contabilização dos votos, após o término das eleições, é um processo fundamental. Os votos são contabilizados mesmo sem serem desencriptados graças à utilização do algoritmo homomórfico ElGamal. Neste contexto, pretende-se avaliar o desempenho do servidor eleitoral quando confrontado com a necessidade de contabilizar os votos submetidos de uma dada eleição. Para a realização do teste definiram-se eleições, através de um *script*, com 10K, 20K, 30K, 40K e 50K votos, um por eleitor, de forma a perceber os tempos médios e o consumo de memória deste processo.

Analisando os dados da tabela 8.6 pode constatar-se que os tempos médios para contabilização dos votos são bastante razoáveis. Todavia, através destes tempos detetou-se um grande consumo de memória à medida que o número de votos crescia e verificou-se que eleições com 50K votos resultavam no erro *HTTP* 500. Este erro ocorreu porque a quantidade de memória necessária ultrapassou o máximo permitido. Assim sendo, procedeu-se à análise do código de forma a identificar potenciais zonas de melhoria. Rapidamente se percebeu que o principal problema consistia no facto de os eleitores, onde consta o último voto, serem carregados para memória. De imediato se percebeu que se teria que resolver este problema e surgiu a ideia de obter apenas, no máximo, 10K eleitores de cada vez. Analisando-se a documentação do Django foi possível verificar que se poderia utilizar um iterador com o $chunk\_size = 10K$. Desta forma, seria possível controlar o aumento de memória ainda que em eleições com mais de 10K votos sejam necessários vários pedidos ao SGBD.

	Tempos de resposta			
Nº Votos eleição	**Min**	**Max**	**Med**	**Memória**
10000	7891	8501	8030	706MB
20000	10035	12005	11050	1.05GB
30000	15040	17032	16080	1.54GB
40000	20010	24008	21050	2.37GB
50000	ERRO 500	ERRO 500	ERRO 500	2.87GB

Tabela 8.6: Tempos de resposta e consumo de memória na contabilização dos votos

Anteriormente identificou-se um problema que degradava o desempenho e sugeriu-se uma possível solução. Neste contexto, apresentam-se agora, através da figura 8.4, as alterações efetuadas. Após estas alterações repetiram-se os testes efetuados anteriormente. Os resultados destes testes podem ser encontrados na tabela 8.7. Nesta, pode verificar-se uma considerável melhoria nos tempos de resposta mas também no consumo de memória que ficou muito mais estável. Com esta melhoria foi possível conseguir-se contabilizar uma eleição com 50K votos. Note-se que os valores obtidos são mais do que suficientes uma vez que os orçamentos participativos, com muito poucas excepções, não têm uma participação nestas ordens de grandeza.

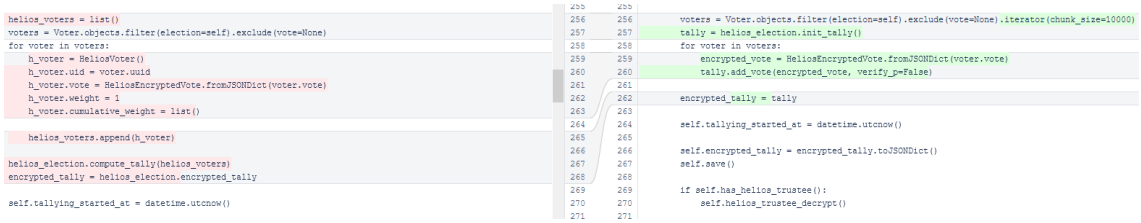

Figura 8.4: Alterações para otimização do processo de contabilização dos votos de uma eleição

	Tempos de resposta			
Nº Votos eleição	**Min**	**Max**	**Med**	**Memória**
10000	5250	5960	5480	604MB
20000	8800	10220	9025	749MB
30000	12880	14460	13210	845MB
40000	16030	17230	16940	850MB
50000	24750	25040	24821	901MB

Tabela 8.7: Tempos de resposta e consumo de memória na contabilização dos votos após otimização

8.2.6 Desencriptar resultado da eleição

Depois dos votos terem sido contabilizados torna-se necessário desencriptar o resultado da eleição. Para isso todos os *trustees* devem facultar as suas desencriptações parciais. Estando todas as desencriptações parciais na posse do servidor, este pode finalmente combinar as mesmas e, consequentemente, revelar o resultado da eleição.

Pretende-se perceber qual é o desempenho do servidor nesta tarefa. Para tal, recorrer-se-á às eleições contabilizadas anteriormente para revelar o resultado. Serão realizados 15

pedidos ao longo de 1 minuto de forma a determinar os valores máximos, mínimos e médios obtidos. Os resultados dos testes evidenciaram que em média desencriptar o resultado da eleição demora cerca de 3.55s tendo-se verificado ainda que no mínimo demoraram 3.48s e no máximo 3.78s o que são resultados muito positivos e que estão dentro dos valores desejáveis.

8.3 Conclusão

No presente capítulo começou-se por abordar alguns dos testes funcionais que foram realizados no contexto do trabalho apresentado nesta dissertação. Estes testes foram fundamentais, e serão também para o futuro, para garantir o correto funcionamento das funcionalidades do servidor eleitoral. Por cada teste aqui apresentado forneceu-se uma breve descrição do que se pretende garantir com o teste, indicou-se o corpo do pedido e, por fim, o resultado expectável.

No que diz respeito aos testes funcionais começou-se por apresentar o serviço utilizado para este tipo de testes, os seus objetivos principais, os diversos tipos de testes disponibilizados pelo Loader.io e o ambiente de testes. Cada teste realizado foi acompanhado com uma breve descrição dos parâmetros utilizados e dos resultados obtidos. Como se pode constar, foi ainda possível detetar e resolver um problema na contabilização dos votos de eleições com mais de 40K votos.

Em suma, os testes apresentados ao longo do capítulo permitiram e permitem garantir o correto funcionamento da solução implementada assim como verificar que os tempos de processamento e resposta às solicitações com o qual o servidor eleitoral é confrontado, estão dentro dos valores aceitáveis e expectáveis.

Capítulo 9

Conclusão

Os orçamentos participativos são mecanismos democráticos importantes que vieram para ficar. São já milhares as implementações deste mecanismo ao redor de todo o mundo. Portugal é um dos países que acolheu e adotou os orçamentos participativos. Dado o acesso cada vez mais massificado a terminais tecnológicos e à internet, e sendo os orçamentos participativos promovidos e votados através de plataformas web que correm nos terminais dos eleitores, torna-se imperativo garantir não só o funcionamento destas plataformas como também a segurança, privacidade e confiabilidade que estas facultam aos eleitores.

Para se conseguir aumentar a segurança dos processos eleitorais, em especial na votação, tornou-se necessário analisar as propostas e contribuições feitas, no sentido de resolver as problemáticas da votação eletrónica, em especial por meios remotos. Assim, nesta dissertação procedeu-se à análise e descrição do Scantegrity II, um sistema *open source* para votação eletrónica presencial, do Helios, um sistema *open source* para votação eletrónica remota a partir da internet, e por fim o EVIV, um sistema para votação eletrónica remota que recorre a VST de forma a garantir a segurança e privacidade mesmo em terminais inseguros.

Da referida análise das propostas, tomou-se a decisão de implementar a *RESTful API* tendo por base o Helios. Tal como referido na subsecção 6.3, tal decisão foi tomada não só, mas essencialmente, pelos seguintes motivos: a) não exigir qualquer equipamento específico ao eleitor, b) não exigir interações complexas aos eleitores (o atual e simples processo de votação pode ser mantido), c) custos de manutenção e infraestrutura reduzidos e d) ser uma solução que permite que o eleitor audite o seu próprio voto e que também outras entidades possam auditar o processo eleitoral. O EVIV, solução teoricamente mais segura, foi considerado mas, por não ter sido uma proposta tão escrutinada pela comunidade e exigir um equipamento e processo mais complexo do que se pretende para os orçamentos participativos, acabou por ser rejeitado.

Ao longo deste trabalho procedeu-se à implementação da *RESTful API* para integrar com as atuais plataformas de orçamento participativo da entidade acolhedora. Esta *API* permite a gestão de todo o processo eleitoral, desde a criação da eleição até à contabilização dos votos, de forma bastante simples e oferece um nível de segurança adequado, face às ameaças geralmente enfrentadas. Desde o início que se procurou implementar uma solução que pudesse ser compatível com o maior número de regras de votação possíveis. A solução proposta e implementada, com a exceção de votos negativos, contempla todas as restantes regras de votação.

O trabalho realizado contribuiu com uma *RESTful API* para a gestão dos processos elei-

torais das plataformas, garantindo a segurança e privacidade do eleitor. Existindo pelo menos um *trustee* honesto, e que preserve a sua chave privada em segurança não a partilhando com ninguém, a solução garante que ninguém consegue desencriptar o voto e ver o seu conteúdo. Adicionalmente, neste trabalho procedeu-se também à alteração do algoritmo de *hashing* SHA1, considerado inseguro, em prol do SHA256 que, aos dias atuais, é considerado seguro e recomendado, inclusive, pelo *National Institute of Standards and Tecnology* (NIST).

De forma a garantir que o sistema se comportava como expectável, ao longo da implementação da *API*, foram-se realizando testes às funcionalidades desenvolvidas. Estes testes comprovaram o correto funcionamento das funcionalidades e, adicionalmente, oferecem maiores garantias se alterações tiverem de ser realizadas, pois se as mesmas causarem algum erro os testes irão falhar e o(s) programador(es) terão que resolver o problema. Para além dos testes referidos, fizeram-se ainda testes de carga/desempenho, para garantir que o sistema responde num intervalo de tempo desejável sob determinadas cargas, mas também para perceber qual é a capacidade que um servidor consegue fornecer. A realização destes testes permitiu descobrir um problema, numa das funcionalidades, com um impacto significativo no desempenho, conforme exposto na secção 8.2.5.

9.1 Trabalho futuro

Como trabalho futuro, identificamos a necessidade de emissão de recibos de votos assinados digitalmente, de forma a que o eleitor tenha consigo uma prova válida do seu voto. O principal objetivo é que o eleitor possa verificar se existe alguma inconformidade no processo, abrir uma disputa e facultar esse recibo como prova para avaliação por parte das autoridades eleitorais.

Uma outra funcionalidade que seria interessante implementar futuramente seria a incorporação da possibilidade de votos negativos. Ainda que esta seja utilizada por um número consideravelmente pequeno de orçamentos participativos, a adição desta funcionalidade completaria ainda mais a implementação.

Dado que a inserção de votos é computacionalmente exigente, o que faz com que rapidamente a capacidade de processamento máxima do CPU seja atingida, consideramos que poderá ser útil e vantajoso implementar, através de um *Task Broker*, uma fila de tarefas, onde são colocados todos os votos submetidos ao servidor para serem processados à medida do possível. Desta forma, mesmo perante eventuais picos de submissão de votos, os pedidos podem ser aceites e, garantidamente, processados gradualmente. Um problema nesta abordagem é que na submissão de eventuais votos inválidos, que não são expectáveis e só devem acontecer por manipulação do corpo do pedido, não seria possível dar essa informação (mensagem a informar que o voto é inválido) na resposta ao pedido.

Referências

[1] Padrões SaaS multi-inquilinos | Microsoft Docs.

[2] SaaS: Single Tenant vs Multi-Tenant - What's the Difference? | Digital Guardian.

[3] Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy.

[4] The Benefits of SaaS Multi-Tenant Architecture | Signiant.

[5] The Benefits of SaaS Multi-Tenant Architecture | Signiant.

[6] Ben Adida. Helios: Web-based Open-audit Voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[7] Abdalla Al-Ameen. E-voting systems vulnerabilities. *IEEE*, 2012.

[8] João Albano Palas Martins Nogueira. *Confiança em Votação Electrónica*. PhD thesis.

[9] Josh Benaloh. Simple verifiable elections. In *Proceedings of the 2006 Electronic Voting Technology Workshop*, June 2006.

[10] Henrique Bruno and Ribeiro Carvalho. A Matemática Via Algoritmo de Criptograa ElGamal † por Glauber Dantas Morais sob orientação do. Technical report, aug 2013.

[11] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[12] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald Rivest, Peter Ryan, Emily Shen, and Alan Sherman. Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes. 2008.

[13] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[14] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. Distributed, end-to-end verifiable, and privacy-preserving internet voting systems. *Computers & Security*, 83:268–299, 2019.

[15] Reginaldo Costa. *Sistema seguro de votação eletrônica multi-cédulas*. PhD thesis, Pontifícia Universidade Católica do Paraná, 2008.

[16] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[17] Simone Fischer-Hbner and Stefan Berthold. Chapter 53 - privacy-enhancing technologies. In John R. Vacca, editor, *Computer and Information Security Handbook (Third Edition)*, pages 759 – 778. Morgan Kaufmann, Boston, third edition edition, 2017.

[18] Craig Gentry. *A fully homomorphic encryption scheme*, volume 20. Stanford university Stanford, 2009.

[19] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 169–178, New York, New York, USA, 2009. ACM Press.

[20] 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, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[21] Dimitris A Gritzalis. Principles and requirements for a secure e-voting system. *Computers & Security*, 21(6):539–556, oct 2002.

[22] Friorik P. Hjalmarsson, Gunnlaugur K. Hreioarsson, Mohammad Hamdaqa, and Gisli Hjalmtysson. Blockchain-Based E-Voting System. In *IEEE International Conference on Cloud Computing, CLOUD*, volume 2018-July, pages 983–986. IEEE Computer Society, sep 2018.

[23] Sherry Hsu. Session vs token based authentication.

[24] Rui Joaquim, Paulo Ferreira, and Carlos Ribeiro. EVIV: An end-to-end verifiable Internet voting system. *Computers & Security*, 32:170–191, feb 2013.

[25] Rui Joaquim and Carlos Ribeiro. An Efficient and Highly Sound Voter Verification Technique and Its Implementation. pages 104–121. 2012.

[26] Rui Joaquim, Carlos Ribeiro, and Paulo Ferreira. VeryVote: A voter verifiable code voting system. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5767 LNCS, pages 106–121. Springer Verlag, 2009.

[27] Ali Kaan Koç, Emre Yavuz, Umut Can Çabuk, and Gökhan Dalkiliç. Towards secure e-voting using ethereum blockchain. In *6th International Symposium on Digital Forensic and Security, ISDFS 2018 - Proceeding*, volume 2018-January, pages 1–6. Institute of Electrical and Electronics Engineers Inc., may 2018.

[28] Nir Kshetri and Jeffrey Voas. Blockchain-Enabled E-Voting. *IEEE Software*, 35(4):95–99, jul 2018.

[29] Andre Kundru. Criptografia homomórfica: um esquema de criptografia cada vez mais usado.

[30] Guy Levin. 4 most used rest api authentication methods.

[31] Sapna Kumari Luiz Thomaz do Nascimento and Vedavinayagam Ganesan. Zero knowledge proofs applied to auctions.

[32] Andreas V Meier. The ElGamal Cryptosystem. Technical report, 2005.

[33] P. Varsha Parmar, Shraddha B. Padhar, Shafika N. Patel, Niyatee I. Bhatt, and Rutvij H. Jhaveri. Survey of various homomorphic encryption algorithms and schemes. *International Journal of Computer Applications*, 91:26–32, 2014.

[34] Thea Peacock, Peter Y.A. Ryan, Steve Schneider, and Zhe Xia. Chapter e90 - verifiable voting systems. In John R. Vacca, editor, *Computer and Information Security Handbook (Third Edition)*, pages e293 – e315. Morgan Kaufmann, Boston, third edition edition, 2013.

[35] Olivier Pereira. Internet Voting with Helios *. Technical report, 2016.

[36] Tiago Rui Carvalho e Pereira. Tecnologias de segurança no e-vote. feb 2006.

[37] Gleudson Pinheiro Varejão Junior. Proposta de um modelo para verificabilidade E2E no sistema eletrônico de votação brasileiro utilizando mecanismos de criptografia visual. aug 2014.

[38] Pedro Vasconcelos and Castro Lopes Faria. Remote Electronic Voting Studying and Improving Helios. Technical report.

Apêndices

Esta página foi deixada em branco propositadamente.

Anexo A - Planeamento do trabalho

Neste anexo apresenta-se o planeamento de trabalhos para o 1º e 2º semestre.

1º Semestre

A planificação dos trabalhos para o 1º semestre foi feita tendo em conta o seguinte plano de trabalhos:

- Familiarização com o estado da arte nomeadamente com os diversos sistemas de votação e os seus requisitos específicos e implementações desses sistemas.
- Familiarização com os processos de votação em sistemas de votação eletrónica.
- Definição dos requisitos que se pretendem garantir.
- Análise de propostas para votação eletrónica.
- Definição de proposta de arquitetura para sistema de votação eletrónica remota.
- Validação da arquitetura.
- Escrita do relatório de intermédio.

2º Semestre

Para o 2º semestre a planificação feita foi realizada tendo em conta o seguinte plano de trabalhos:

- Familiarização com as primitivas criptográficas geralmente utilizadas para votação eletrónica remota.
- Estudo e escolha de uma proposta para votação eletrónica remota.
- Implementação de uma RESTful API baseada na proposta escolhida.
- Validação da solução.
- Escrita do relatório final.

Esta página foi deixada em branco propositadamente.

Anexo B - R3 - Eliminação de eleição

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- A eleição não pode estar congelada (*freezed*).
- O cliente só pode eliminar eleições criadas por si.

Funcionamento:

Através de um pedido DELETE os clientes da API podem proceder à eliminação de uma eleição, do próprio cliente, desde que a eleição não esteja congelada. Todos os *trustees* desta eleição são também eliminados como consequência.

Diagrama de sequência

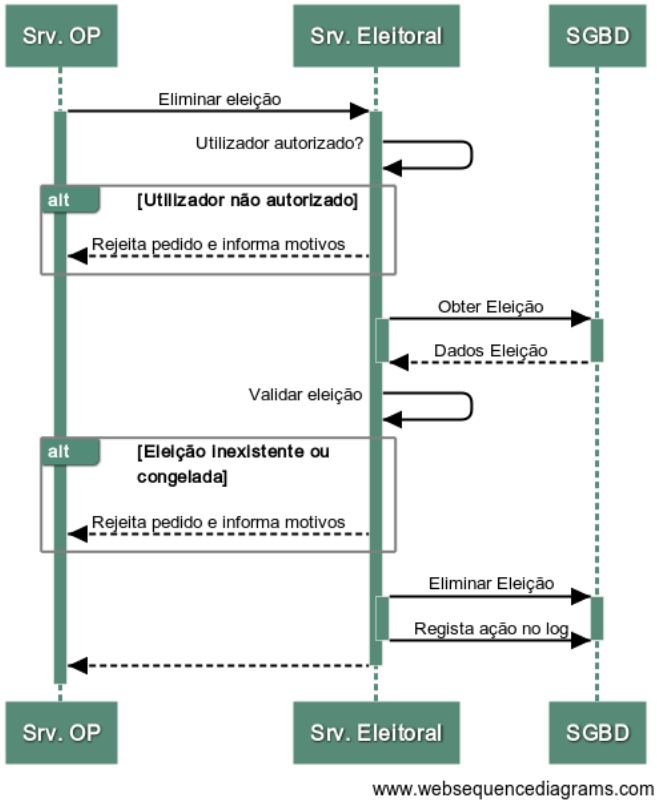

Diagrama sequência eliminação de eleição

Esta página foi deixada em branco propositadamente.

Anexo C - R4 - Lista de eleições

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode ver eleições criadas por si.

Funcionamento:

Através de um pedido GET os clientes da API podem obter uma lista, paginada, com as suas eleições. O funcionamento desta funciona da seguinte forma:

1. Ao receber o pedido a API consulta a base de dados sobre as eleições do cliente autenticado. No pedido podem ser facultados a página e o numero de elementos por página a obter.
2. Ao receber os dados da base de dados a API irá proceder à sanitização de alguns dos atributos, nomeadamente o título e descrição, de cada eleição de forma a prevenir a exploração de XSS (como ditam as boas práticas de programação segura do OWASP). O processo de sanitização é realizado recorrendo à invocação do método "clean" da biblioteca bleach [1].
3. Por fim, os dados sanitizados são retornados ao cliente.

Diagrama de sequência

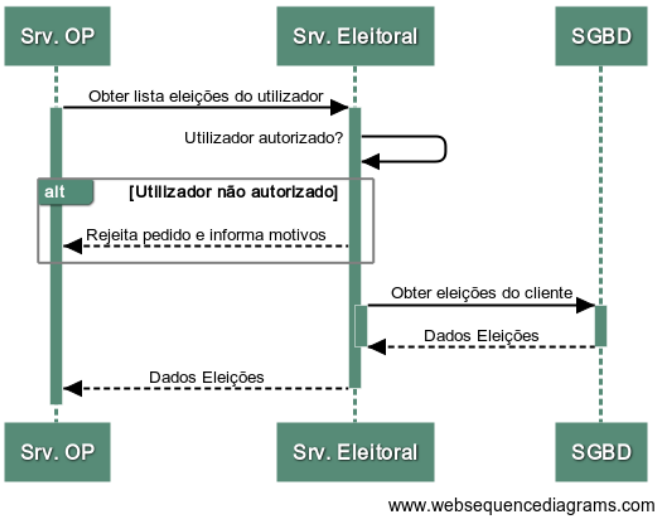

Diagrama sequência lista de eleições

[1] Biblioteca de sanitização de HTML para Python que pode ser encontrada em https://pypi.org/project/bleach/

Esta página foi deixada em branco propositadamente.

Anexo D - R5 - Detalhe eleição

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode ver os detalhes de uma eleição criada por si.

Funcionamento:

Através de um pedido GET os clientes da API podem obter os detalhes da eleição. Através do identificador da eleição, que consta no url, é verifica-se a existência desta eleição. Adicionalmente,verifica-se também se esta eleição é da autoria do cliente autenticado e, se tal não se verificar, o pedido é rejeitado com o código 403 [2] e o motivo por detrás da rejeição (no caso, não ser uma eleição de sua autoria). Por outro lado, se estiver tudo bem, o servidor irá retornar um conjunto definido de atributos da eleição.

Diagrama de sequência

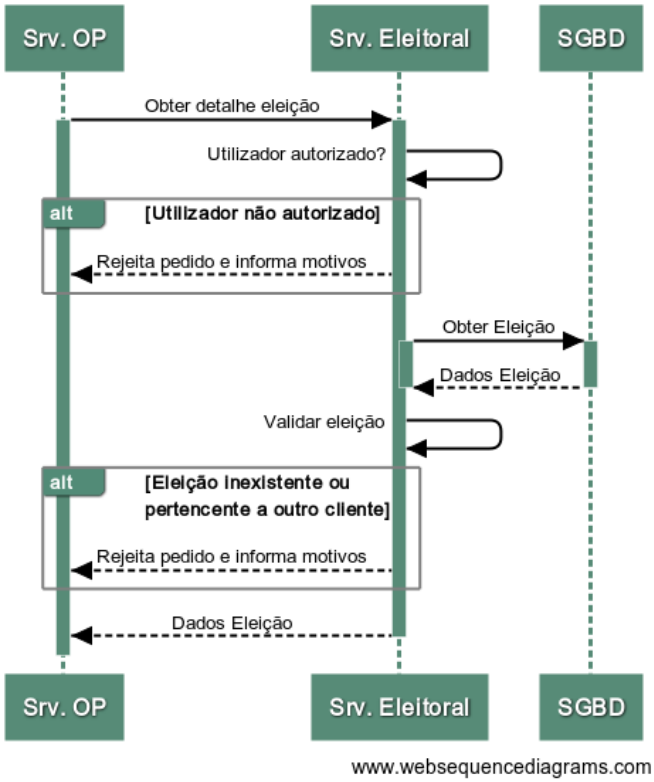

Diagrama sequência detalhe eleição

[2]Código de estado do definido no RFC 7231 que indica que o servidor recebeu o pedido, reconheceu a identidade do utilizador mas não autorizou o processamento do mesmo.

Esta página foi deixada em branco propositadamente.

Anexo E - R7 - Eliminar eleitor

Classificação: Must (Essencial)

Prioridade: Alta

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode eliminar eleitores em eleições criadas por si.

Funcionamento:

Através de um pedido DELETE os clientes da API podem eliminar eleitores de uma eleição sua. A eliminação dos eleitores só pode ser feita se, e apenas se, forem cumpridas as seguintes condições:

- **Tratando-se de uma eleição aberta** a eliminação pode ser feita se não existirem votos deste eleitor e a eleição não estiver arquivada. OU
- Até ao **congelamento da eleição**.

Caso as referidas condições sejam satisfeitas o eleitor é removido da base de dados e é retornada uma mensagem de sucesso. Caso contrário, o pedido é rejeitado com o estado 403 e são retornados os motivos pelo qual não é possível proceder à eliminação do dado eleitor.

Diagrama de sequência

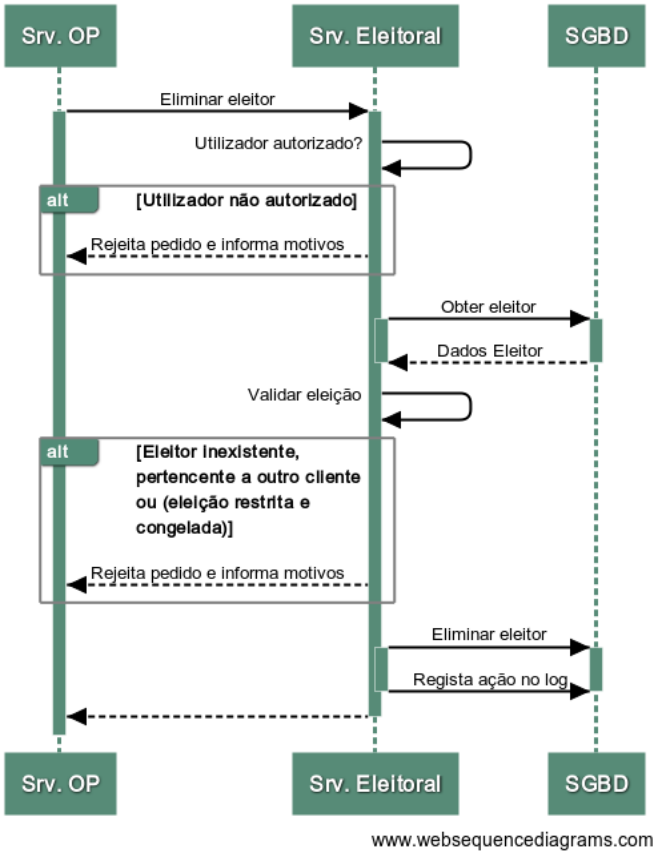

Diagrama sequência eliminação de eleitor

Esta página foi deixada em branco propositadamente.

Anexo F - R10 - Obter lista trustees

Classificação: Must (Essencial)

Prioridade: Média

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode obter a lista de *trustees* de uma eleição criada por si.

Funcionamento:

Através de um pedido GET os clientes da API podem obter a lista de todos os *trustees* da eleição. Destes são retornados o nome e a chave pública armazenadas na base de dados.

Diagrama de sequência

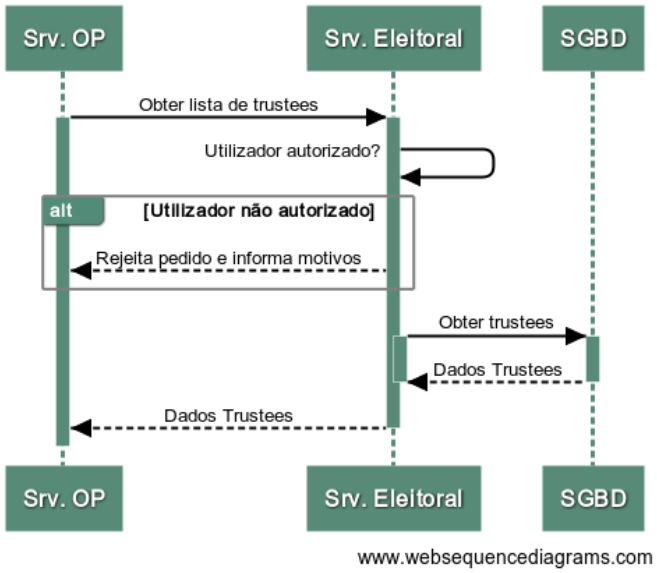

Diagrama sequência lista trustees

Esta página foi deixada em branco propositadamente.

Anexo G - R12 - Estender período de votação

Classificação: Should

Prioridade: Média

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode entender o período eleitoral de eleições criadas por si.
- A eleição tem de estar congelada (freezed) e não estar arquivada.

Funcionamento:

A extensão do prazo de uma eleição é possível através de um pedido com o método PUT. Neste pedido deve ser é facultado o identificador da eleição e a nova data de término da eleição que é então definida na eleição através do atributo "voting_extended_until". Se a atualização tiver sucesso é retornada uma mensagem de sucesso. Caso contrário é retornada uma mensagem de erro genérica.

Diagrama de sequência

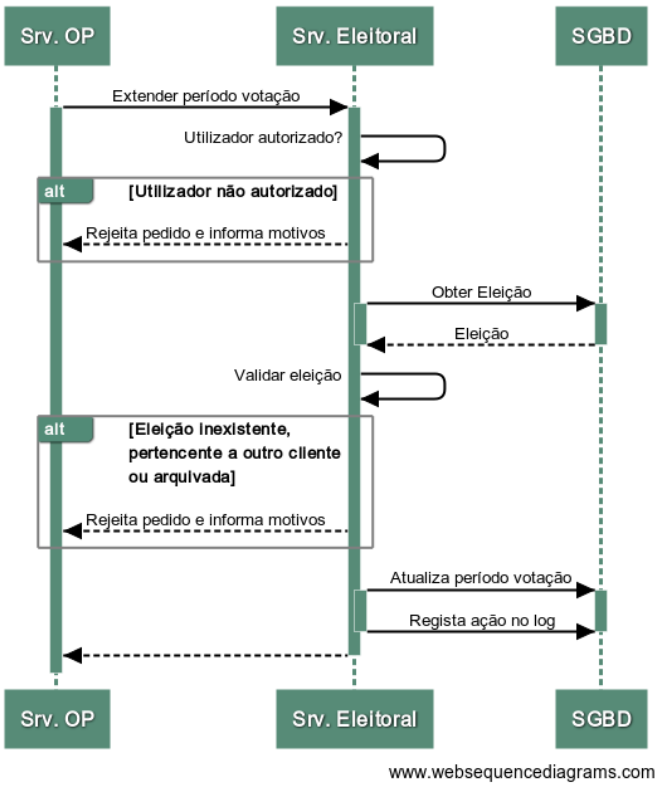

Diagrama sequência estender período votação

Esta página foi deixada em branco propositadamente.

Anexo H - R13 - Arquivar eleição

Classificação: Should

Prioridade: Baixa

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode arquivar eleições criadas por si.

Funcionamento:

O processo de arquivamento de uma eleição consiste em apenas definir o atributo "archived_at" com a data e hora do momento do pedido. Se a atualização da eleição tiver sucesso é retornada uma mensagem de sucesso. Caso contrário é retornada uma mensagem de erro genérica.

Diagrama de sequência

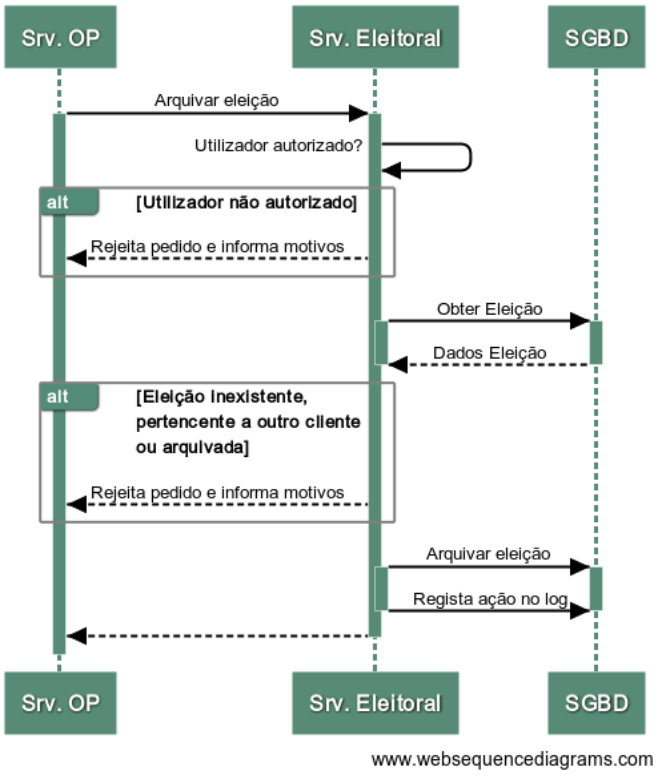

Diagrama sequência arquivar eleição

Esta página foi deixada em branco propositadamente.

Anexo I - R20 - Obter lista de votos

Classificação: Must

Prioridade: Média

Estado: Concluído

Restrições:

- Autenticado como gestor de eleições.
- O cliente só pode obter a de votos de uma eleição criada por si.

Funcionamento:

Através de um pedido GET os clientes da API podem obter a lista de todos os votos submetidos ao servidor eleitoral. Destes é retornado o nome ou*alias* do eleitor e o *hash* do voto.

Esta página foi deixada em branco propositadamente.

Anexo J

Código teste descrito na subsecção 8.1.1.

```
def test_election_mandatory_parameters_are_properly_set(self):
    data = {}
    response = self.client.post('/elections', data=data, format='json')
    required_field_message = ErrorDetail('This field is required.', code='required')
    expected_result = {'short_name': [required_field_message], 'name': [required_field_message],
                       'description': [required_field_message]}
    self.assertEquals(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEquals(response.data, expected_result)
```

Código teste descrito na subsecção 8.1.2.

```
def test_election_can_be_created_under_valid_parameters(self):
    data = {
        'short_name': 'OP2020',
        'name': 'Orcamento Participativo 2020',
        'description': 'Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...'
    }
    expected_keys = {'uuid', 'short_name', 'name', 'description', 'election_type', 'createdAt',
                     'lastModifiedAt', 'lastModifiedAt', 'archived_at', 'frozen_at', 'public_key', 'questions',
                     'use_voter_aliases', 'use_advanced_audit_features', 'randomize_answer_order', 'openreg',
                     'voters_hash', 'voting_extended_until', 'registration_starts_at', 'voting_starts_at',
                     'voting_ends_at', 'voting_started_at', 'voting_ended_at', 'tallying_started_at',
                     'tallying_finished_at', 'tallies_combined_at', 'result_released_at', 'encrypted_tally',
                     'result', 'result_proof'}
    keys_with_expected_none_value = {
        'archived_at', 'frozen_at', 'public_key', 'voters_hash', 'voting_ends_at', 'voting_started_at',
        'voting_ended_at', 'tallying_started_at', 'tallying_finished_at', 'tallies_combined_at',
        'result_released_at', 'encrypted_tally', 'result', 'result_proof'
    }
    response = self.client.post('/elections', data=data, format='json')
    self.assertEquals(response.status_code, status.HTTP_201_CREATED)
    # Check if the keys returned are the same, independently of the order.
    self.assertCountEqual(response.data.keys(), expected_keys)
    self.assertEquals(response.data['public_key'], None)
    # Check if, by default, the use_voter_aliases is set to False
    self.assertFalse(response.data['use_voter_aliases'])
    # Check if, by default, the use_voter_aliases is set to True
    self.assertTrue(response.data['use_advanced_audit_features'])
    # Check if, by default, the randomize_answer_order is set to False
    self.assertFalse(response.data['randomize_answer_order'])
    # Check if, by default, the openreg is set to False
    self.assertFalse(response.data['openreg'])
    # Check if HeliosTrustees was created and their private and public keys are defined
    try:
        helios_trustee = Trustee.objects.get(election=response.data['uuid'])
    except Trustee.DoesNotExist:
        self.fail('HeliosTrustee was not created')

    self.assertNotEquals(helios_trustee.secret, None)
    self.assertNotEquals(helios_trustee.public_key, None)
    self.assertNotEquals(helios_trustee.public_key_hash, None)
    self.assertNotEquals(helios_trustee.secret_key, None)
```

Código teste descrito na subsecção 8.1.3.

```
def test_trustee_can_be_added_under_valid_parameters(self):
    election_data = self.create_election_ready_for_voting()
    election_uuid = election_data['uuid']
    # Generate the trustee cryptografic stuff.

    helios_trustee = generate_helios_trustee()

    data = {
        'name': 'Leandro',
        'public_key': helios_trustee.public_key.toJSONDict(),
        'pok': helios_trustee.pok.toJSONDict()
    }

    expected_keys = {'uuid', 'name', 'public_key', 'pok', 'public_key_hash', 'election'}
    response = self.client.post('/elections/' + election_uuid + '/trustees', data=data, format='json')
    self.assertEquals(response.status_code, status.HTTP_201_CREATED)
    # Check if the keys returned are the same, independently of the order.
    self.assertCountEqual(response.data.keys(), expected_keys)
    self.assertNotEquals(response.data['public_key'], None)
    self.assertNotEquals(response.data['pok'], None)
    self.assertNotEquals(response.data['election'], None)
    self.assertNotEquals(response.data['public_key_hash'], None)
```

Código teste descrito na subsecção 8.1.4.

```
def test_voter_can_be_added_under_valid_parameters(self):
    election_data = self.create_election_ready_for_voting()
    election_uuid = election_data['uuid']

    data = {
        'election': election_uuid,
        'voter_id': 'V3',
        'name': 'Leandro'
    }
    expected_keys = {'uuid', 'voter_id', 'name', 'alias', 'vote', 'vote_hash', 'cast_at', 'election'}
    keys_with_expected_none_value = {'alias', 'vote', 'vote_hash', 'cast_at'}
    response = self.client.post('/voters', data=data, format='json')
    self.assertEquals(response.status_code, status.HTTP_201_CREATED)
    # Check if the keys returned are the same, independently of the order.
    self.assertCountEqual(response.data.keys(), expected_keys)

    # Check if some keys have the None as a value.
    errors = []
    for key in keys_with_expected_none_value:
        if response.data[key] is not None:
            errors.append('The ' + key + ' is expected to be None but we got ' + response.data[key])
    self.assertEquals(len(errors), 0, errors)
```

Código teste descrito na subsecção 8.1.5.

```
def test_cant_add_two_voters_with_same_id(self):
    election_data = self.create_election_ready_for_voting()
    election_uuid = election_data['uuid']

    dataV1 = {
        'election': election_uuid,
        'voter_id': 'V3',
        'name': 'Leandro'
    }
    dataV2 = {
        'election': election_uuid,
        'voter_id': 'V3',
        'name': 'Rodrigues'
    }
    responseV1 = self.client.post('/voters', data=dataV1, format='json')
    self.assertEquals(responseV1.status_code, status.HTTP_201_CREATED)
    responseV2 = self.client.post('/voters', data=dataV2, format='json')
    self.assertEquals(responseV2.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertTrue('non_field_errors' in responseV2)
    self.assertEquals(responseV2.data['non_field_errors'], ErrorDetail('The fields election, voter_id must make a unique set.', 'unique'))
```

Código teste descrito na subsecção 8.1.6.

```
def create_election_ready_for_voting(self, open_reg=True):...

def freeze_election(self, election_uuid):
    response = self.client.put('/elections/' + election_uuid + '/freeze')
    self.assertEquals(response.status_code, status.HTTP_200_OK)
    self.assertNotEquals(response.data['public_key'], None)
    self.assertNotEquals(response.data['frozen_at'], None)
    return response

def test_freezing_election_sets_public_key_of_election(self):
    election_data = self.create_election_ready_for_voting()
    # Let's freeze the election now
    self.freeze_election(election_data['uuid'])
```

Código teste descrito na subsecção 8.1.7.

```
def cast_vote(self, election_uuid, voter_uuid, plain_ballot):
    response = self.client.post('/elections/' + election_uuid + '/encrypt_ballot', data={'ballot': plain_ballot},
                                format='json')
    self.assertEquals(response.status_code, status.HTTP_200_OK)
    encrypted_vote = response.data

    # Now we have the encrypted vote it's time to cast the vote
    vote_data = {'encrypted_vote': encrypted_vote, 'voter_uuid': voter_uuid}
    response = self.client.post('/elections/' + election_uuid + '/cast_vote', data=vote_data, format='json')
    self.assertEquals(response.status_code, status.HTTP_201_CREATED)
    self.assertNotEquals(response.data['vote_hash'], None)
    return response

def test_add_encrypted_vote_successfully(self):
    election_data = self.create_election_ready_for_voting()
    election_uuid = election_data['uuid']

    # Let's freeze the election now
    self.freeze_election(election_uuid)

    # We already have the election freezed. It's now time
    # to add a voter to the election. This is only possible after
    # frozen the election because it's a public election (openreg=True)

    response = self.add_voter_to_election(election_uuid, 'Leandro', 'V3')
    voter_uuid = response.data['uuid']

    # Now we have a voter already registered let's try to add an encrypted vote.
    # In this test we will use server encryption. In production the votes will be encrypted
    # at their own browsers using the same algorithm as the server.

    plain_ballot = [[1]]
    self.cast_vote(election_uuid, voter_uuid, plain_ballot)
```

Código teste descrito na subsecção 8.1.9.

```
def compute_tally(self, election_uuid):
    response = self.client.post('/elections/' + election_uuid + '/compute_tally', data={}, format='json')
    self.assertEquals(response.status_code, status.HTTP_200_OK)
    self.assertTrue('ready_for_decryption_combination' in response.data)
    self.assertTrue('encrypted_tally' in response.data)
    self.assertNotEquals(response.data['encrypted_tally'], None)
    self.assertTrue(response.data['ready_for_decryption_combination'])
    return response

def test_compute_tally(self):
    election_data = self.create_election_ready_for_voting()
    election_uuid = election_data['uuid']

    # Let's freeze the election now
    self.freeze_election(election_uuid)

    # We already have the election freezed. It's now time
    # to add a voter to the election. This is only possible after
    # frozen the election because it's a public election (openreg=True)

    response = self.add_voter_to_election(election_uuid, 'Leandro', 'V1')
    voter1_uuid = response.data['uuid']
    response = self.add_voter_to_election(election_uuid, 'Rodrigues', 'V2')
    voter2_uuid = response.data['uuid']

    # Now we have a voter already registered let's try to add the encrypted votes.
    # In this test we will use server encryption. In production the votes will be encrypted
    # at their own browsers using the same algorithm as the server.

    plain_ballot1 = [[1]]
    plain_ballot2 = [[0]]

    self.cast_vote(election_uuid, voter1_uuid, plain_ballot1)
    self.cast_vote(election_uuid, voter2_uuid, plain_ballot2)

    self.compute_tally(election_uuid)
```

Código teste descrito na subsecção 8.1.10.

```
def test_combine_decryptions(self):
    election_data = self.create_election_ready_for_voting()
    election_uuid = election_data['uuid']

    # Let's freeze the election now
    self.freeze_election(election_uuid)

    # We already have the election freezed. It's now time
    # to add a voter to the election. This is only possible after
    # frozen the election because it's a public election (openreg=True)

    response = self.add_voter_to_election(election_uuid, 'Leandro', 'V1')
    voter1_uuid = response.data['uuid']
    response = self.add_voter_to_election(election_uuid, 'Rodrigues', 'V2')
    voter2_uuid = response.data['uuid']

    # Now we have a voter already registered let's try to add the encrypted votes.
    # In this test we will use server encryption. In production the votes will be encrypted
    # at their own browsers using the same algorithm as the server.

    plain_ballot1 = [[0]]
    plain_ballot1_override = [[1]]
    plain_ballot2 = [[1]]

    self.cast_vote(election_uuid, voter1_uuid, plain_ballot1)
    self.cast_vote(election_uuid, voter2_uuid, plain_ballot2)
    self.cast_vote(election_uuid, voter1_uuid, plain_ballot1_override)  # Only this vote should be counted fir voter1

    self.compute_tally(election_uuid)

    response = self.client.post('/elections/' + election_uuid + '/combine_decryptions', data={}, format='json')
    self.assertEquals(response.status_code, status.HTTP_200_OK)
    self.assertTrue('result' in response.data)
    self.assertEquals(response.data['result'], [[0, 2]])
    return response.data
```

Esta página foi deixada em branco propositadamente.

Anexo K - Interface criação de teste no Loader.io