Gabriel Kusumota Nadalin

E2Easy-PC: Adaptação e Implementação em Rust do Protocolo E2Easy com Compromissos de Pedersen

São Carlos - SP
2025

Gabriel Kusumota Nadalin

E2Easy-PC: Adaptação e Implementação em Rust do Protocolo E2Easy com Compromissos de Pedersen

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Paulo Matias
Coorientador: Leonardo Toshinobu Kimura

Universidade Federal de São Carlos – UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Computação – DC
Bacharelado em Ciência da Computação

São Carlos - SP
2025

Dedico este trabalho ao Gemini e ao Jules, as IAs que tornaram este trabalho possível (e tolerável). Sem vocês, eu ainda estaria escrevendo a primeira linha de texto.

Agradecimentos

Agradeço aos meus orientadores, à minha família, aos meu amigos, às minhas gatas (2), à minha namorada (3D), ao Stack Overflow, à documentação do Rust, e à minha máquina por não travar durante a compilação. Um agradecimento especial ao café, combustível essencial desta jornada.

"Na minha máquina funciona."

Autor Desconhecido

Resumo

O presente trabalho apresenta o E2Easy-PC, uma adaptação do protocolo de votação E2Easy utilizando Compromissos de Pedersen e implementação segura em Rust. Enquanto o protocolo original propõe o uso de reticulados (Lattices) para segurança pós-quântica completa, esta variante adota uma abordagem pragmática focada em *everlasting privacy* (privacidade eterna) e segurança de memória. Substituímos os compromissos baseados em Lattices por Compromissos de Pedersen, que oferecem *perfect hiding* (ocultação perfeita), garantindo que o sigilo do voto jamais seja quebrado, mesmo por computadores quânticos futuros. A implementação em Rust mitiga vulnerabilidades críticas de memória, atendendo a recomendações modernas de segurança (NSA). Os resultados experimentais demonstram que a abordagem com curvas elípticas supera o desempenho da original em ordens de magnitude nas fases de votação e embaralhamento, enquanto o paralelismo de dados reduziu o tempo de apuração em mais de 50%. O trabalho consolida uma arquitetura modular que alia eficiência operacional, segurança de software e privacidade perpétua.

Palavras-chave: Votação Eletrônica. Verificabilidade Fim-a-Fim. Rust. *Mixnets*. E2Easy. Privacidade Eterna.

Abstract

This work presents E2Easy-PC, an adaptation of the E2Easy voting protocol using Pedersen Commitments and secure implementation in Rust. While the original protocol proposes the use of lattices for full post-quantum security, this variant adopts a pragmatic approach focused on Everlasting Privacy and memory safety. We replaced Lattice-based commitments with Pedersen Commitments, which offer Perfect Hiding, ensuring that vote secrecy is never broken, even by future quantum computers. The Rust implementation mitigates critical memory vulnerabilities, complying with modern software security recommendations (NSA). Experimental results demonstrate that the elliptic curve approach outperforms the original by orders of magnitude in the voting and shuffling phases, while data parallelism reduced tallying time by over 50%. The work consolidates a modular architecture that combines operational efficiency, software security, and perpetual privacy.

Keywords: Electronic Voting. End-to-End Verifiability. Rust. Mixnets. E2Easy. Everlasting Privacy.

Lista de abreviaturas e siglas

E2E	End-to-End (Fim-a-Fim)
RDV	Registro Digital do Voto
RDV'	RDV Embaralhado (votos às claras desvinculados)
RDCV	Registro Digital do Compromisso do Voto
RDCV'	RDCV Embaralhado
ZKP	Zero-Knowledge Proof (Prova de Conhecimento Zero)
RLA	Risk-Limiting Audits (Auditorias de Risco)
EAC	Election Assistance Commission
PQC	Post-Quantum Cryptography (Criptografia Pós-Quântica)
DRE	Direct Recording Electronic (Registro Eletrônico Direto)
ECDSA	Elliptic Curve Digital Signature Algorithm
NIST	National Institute of Standards and Technology

Sumário

1 Introdução

A integridade do processo eleitoral é um pilar fundamental das democracias modernas. A transição dos sistemas de votação baseados em papel para sistemas eletrônicos, iniciada nas últimas décadas, trouxe benefícios inegáveis em termos de logística e rapidez na apuração. No Brasil, a adoção de urnas eletrônicas do tipo DRE (*Direct Recording Electronic*) eliminou fraudes comuns no processo manual e acelerou a consolidação dos resultados. No entanto, a natureza "caixa preta" desses dispositivos — onde o registro e a contagem dos votos ocorrem internamente no software, sem um rastro físico auditável pelo eleitor — impõe desafios significativos à transparência e à confiança pública (1).

Para mitigar essas limitações, a comunidade acadêmica desenvolveu o conceito de Verificabilidade Fim-a-Fim (E2E - *End-to-End Verifiability*). Sistemas E2E permitem que o eleitor verifique se seu voto foi registrado conforme sua intenção (*cast-as-intended*) e fornecem provas matemáticas de que todos os votos registrados foram corretamente contabilizados no resultado final (*recorded-as-cast* e *tallied-as-recorded*) (2, 3).

1.1 Definição do Problema

A concepção de sistemas de votação eletrônica segura envolve um equilíbrio delicado entre duas propriedades conflitantes: integridade (o resultado reflete a vontade dos eleitores) e privacidade (o voto não pode ser associado ao eleitor).

A maioria das soluções E2E consolidadas na literatura, como Helios (4), Belenios (5) e implementações baseadas no ElectionGuard (3), fundamenta sua segurança na criptografia de chave pública clássica, especificamente na dificuldade do problema do Logaritmo Discreto (em grupos multiplicativos ou curvas elípticas). Nesses sistemas, a privacidade do voto é *computacional*: o sigilo é mantido apenas enquanto for computacionalmente inviável resolver o problema matemático subjacente.

Com o avanço da computação quântica, essa premissa torna-se vulnerável. O algoritmo de Shor, se executado em um computador quântico suficientemente potente, pode quebrar a criptografia baseada em Logaritmo Discreto e Fatoração, permitindo a decifração retroativa de votos passados. Para protocolos que dependem exclusivamente dessas premissas para o sigilo, isso ameaça a propriedade de *Everlasting Privacy* (Privacidade Eterna) (6): a garantia de que o sigilo do voto será preservado perpetuamente, independentemente da evolução tecnológica.

O protocolo E2Easy original (7) propõe o uso de criptografia baseada em reticulados (*Lattices*) para oferecer segurança pós-quântica. Embora essa abordagem proteja a integridade contra adversários quânticos futuros, a parametrização escolhida para os esquemas de compromisso (como BDLOP) priorizou a eficiência, resultando em ocultação computacional

(*computational hiding*). Embora seja teoricamente possível parametrizar reticulados para obter ocultação estatística (e, portanto, privacidade eterna), tal configuração implicaria em custos de armazenamento e processamento significativamente maiores, o que foi evitado na proposta original em favor do desempenho. Consequentemente, naquela configuração específica, a privacidade dos eleitores ainda poderia estar em risco a longo prazo se as suposições de dureza dos reticulados (LWE/SIS) fossem quebradas.

Além dos desafios criptográficos, a segurança da implementação de software é crítica. Implementações de referência de protocolos criptográficos são frequentemente escritas em C ou C++ para desempenho, linguagens que não oferecem garantias de segurança de memória. Vulnerabilidades como *buffer overflows* e *use-after-free* permanecem como vetores primários de ataques cibernéticos, conforme alertado por agências de segurança como a NSA (8).

1.2 Objetivos

Este trabalho tem como objetivo geral desenvolver e avaliar o **E2Easy-PC**, uma adaptação do protocolo de votação E2Easy focada na garantia de Privacidade Eterna e na segurança de software.

Os objetivos específicos são:

1. **Adaptação Criptográfica**: Substituir as primitivas baseadas em reticulados do protocolo original por Compromissos de Pedersen sobre Curvas Elípticas. Essa escolha estratégica sacrifica a integridade pós-quântica (necessária apenas durante a eleição) para obter *Perfect Hiding* (necessária perpetuamente), garantindo que o sigilo do voto seja matematicamente inviolável, mesmo por poder computacional infinito.

2. **Implementação Segura**: Implementar o protocolo adaptado utilizando a linguagem Rust, assegurando *memory safety* em tempo de compilação e eliminando classes de vulnerabilidades críticas presentes em implementações legadas.

3. **Avaliação de Desempenho**: Analisar o impacto da troca de primitivas na eficiência do sistema e demonstrar a viabilidade prática da solução através do uso de paralelismo de dados na geração das provas de embaralhamento.

1.3 Contribuições

As principais contribuições deste trabalho incluem:

- Uma análise do *trade-off* de segurança entre integridade pós-quântica e privacidade eterna no contexto de votação eletrônica, justificando a escolha por Compromissos de Pedersen.

- A implementação em Rust do protocolo, cujo código-fonte está disponível publicamente em <https://github.com/gabriel-nadalin/E2Easy-PC>, servindo como referência para o desenvolvimento de *mixnets* verificáveis seguras e performáticas.
- Resultados experimentais que demonstram que, embora a segurança pós-quântica completa seja desejável, soluções baseadas em curvas elípticas ainda oferecem desempenho superior e garantias de privacidade mais fortes a longo prazo, sendo uma alternativa pragmática para o cenário atual.

1.4 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma: O Capítulo 2 apresenta a revisão da literatura sobre sistemas de votação e o estado da arte. O Capítulo 3 estabelece a fundamentação teórica necessária. O Capítulo 4 descreve o protocolo E2Easy e suas estruturas. O Capítulo 5 detalha as adaptações realizadas e a análise de segurança. O Capítulo 6 descreve a implementação em Rust. O Capítulo 7 apresenta a avaliação de desempenho e o Capítulo 8 conclui o trabalho.

1.5 Declaração de Uso de IA

A elaboração deste trabalho contou com o auxílio de ferramentas de inteligência artificial generativa como suporte instrumental. Utilizaram-se o modelo Google Gemini 3 Pro e a plataforma de agente Jules, especificamente para: (1) elaboração de esboços para organização do texto e (2) revisão ortográfica, gramatical e de estilo.

É imperativo ressaltar que todo o conteúdo gerado por essas ferramentas foi submetido a uma rigorosa validação humana, assegurando a precisão factual e a aderência à literatura especializada. A concepção intelectual, a análise crítica, a síntese dos dados e a redação final permanecem de inteira responsabilidade do autor, que validou integralmente o material produzido.

2 Revisão da Literatura

2.1 Panorama e Objetivos

Os sistemas de votação digital segura buscam garantir três pilares fundamentais. A *verificabilidade fim-a-fim* permite que eleitores e auditores independentes verifiquem se os votos foram *cast-as-intended* (lançados como pretendido), *recorded-as-cast* (gravados como lançados) e *tallied-as-recorded* (apurados como gravados), e se o resultado corresponde ao conjunto de votos publicados (3, 2). A *privacidade e o anonimato* asseguram a dissociação entre a identidade do eleitor e o conteúdo do voto, oferecendo resistência à coerção e à ligação de voto, sendo *mixnets* e criptografia homomórfica as abordagens centrais para atingir essas garantias (6, 9). Por fim, a *auditabilidade operacional* é obtida por meio da publicação de artefatos verificáveis — boletins, provas criptográficas, hashes e assinaturas — complementada pelo uso de auditorias de risco (RLA) (10).

No contexto brasileiro, a votação eletrônica ocorre em urnas isoladas (*air-gapped*) que operam como terminais de registro eletrônico direto (DRE), sem conexão de rede durante o processo eleitoral.

Esta revisão examina soluções E2E consolidadas internacionalmente, identificando suas características arquiteturais, garantias criptográficas e limitações operacionais, para então posicionar a proposta E2Easy-PC desenvolvida neste trabalho.

2.2 Helios

Helios é um sistema de votação web de código aberto que emprega criptografia homomórfica (ElGamal) e provas de conhecimento zero para obter verificabilidade E2E (4). Os votos são cifrados no cliente, e o escrutínio é realizado por *threshold decryption* (decifração conjunta) ou por provas de correção da contagem. O sistema oferece simplicidade de uso e implantação, sendo amplamente utilizado em ambientes acadêmicos e organizacionais de pequeno a médio porte, com interface web intuitiva e publicação de boletins criptográficos auditáveis.

Limitações: A privacidade baseia-se na segurança computacional do logaritmo discreto (vulnerável a avanços futuros). O sistema é vulnerável a ameaças em clientes comprometidos (extração de chaves, *cold-boot attacks*) e oferece resistência limitada à coerção, pois recibos de verificação podem ser utilizados para pressão sobre eleitores.

2.3 ElectionGuard

ElectionGuard é um conjunto de bibliotecas e especificações desenvolvido pela Microsoft para eleições E2E com publicação de dados auditáveis, cifradores homomórficos (ElGamal) e provas criptográficas associadas (3). O framework é projetado para interoperabilidade com sistemas de votação existentes, oferecendo um ecossistema completo com ferramentas de verificação, formatos padronizados de publicação e suporte a auditorias independentes. Posições institucionais sobre E2E são documentadas por órgãos reguladores como a EAC (U.S. Election Assistance Commission) (2).

Limitações: A complexidade operacional é significativa, exigindo cerimônias criptográficas para geração e destruição de chaves distribuídas (*key ceremonies*), governança rigorosa e infraestrutura de suporte especializada. Como o sistema emprega criptografia homomórfica ElGamal, a privacidade é computacional, não oferecendo proteção perpétua contra adversários futuros. Além disso, o ElectionGuard utiliza soma homomórfica para a totalização, o que limita sua aplicabilidade a eleições com poucos candidatos, sendo inadequado para pleitos com centenas ou milhares de opções como ocorre no Brasil.

2.4 Outras soluções E2E relevantes

Além dos sistemas descritos acima, a literatura apresenta soluções especializadas que abordam contextos operacionais específicos ou exploram arquiteturas híbridas. Examinamos três exemplos representativos: CHVote (votação remota com canais auxiliares), Scantegrity (integração com papel óptico) e Belenios (evolução do Helios).

2.4.1 CHVote (Suíça)

CHVote é um sistema de votação online E2E desenvolvido para cantões suíços, combinando criptografia homomórfica com *mixnets* e suportando múltiplos dispositivos de verificação (11). A verificação individual é implementada através de códigos impressos distribuídos previamente pelo correio, que o eleitor utiliza para confirmar que seu voto foi registrado corretamente. Após a votação, os votos cifrados são processados por uma *mixnet* com re-cifração antes da decifração, e artefatos criptográficos completos são publicados para verificação universal.

Limitações: A logística de distribuição prévia dos códigos de verificação é complexa e custosa, sendo inadequada para o modelo de votação presencial brasileiro, onde o eleitor comparece apenas no dia da eleição sem materiais prévios.

2.4.2 Scantegrity

Scantegrity é um sistema E2E para votação por papel óptico (*optical scan*) que utiliza tinta invisível e códigos de confirmação únicos por candidato (12). O sistema preserva a interface familiar de cédulas de papel: eleitores marcam suas escolhas convencionalmente, e códigos de confirmação são revelados através de tinta especial no momento da marcação. A cédula física serve como registro auditável, enquanto códigos publicados online permitem verificação individual.

Limitações: Dependência crítica de impressão especializada (tinta invisível, códigos únicos por cédula), custos elevados de produção e dificuldades de escalabilidade. O sistema não se aplica ao modelo DRE puramente eletrônico.

2.4.3 Belenios

Belenios é um sistema de votação web com verificabilidade E2E desenvolvido pelo INRIA francês, baseado no protocolo Helios-C (5). Projetado para aprimorar a segurança e a usabilidade do Helios original, o sistema utiliza criptografia homomórfica (ElGamal) e provas de conhecimento zero, sendo adequado para eleições de pequeno a médio porte (assembleias, conselhos universitários, eleições organizacionais).

Limitações: Como descendente direto do Helios, compartilha das limitações de privacidade computacional (não perpétua) inerentes ao esquema ElGamal.

2.4.4 Adaptações Pós-Quânticas

Recentemente, propostas têm surgido para adaptar sistemas clássicos ao cenário pós-quântico. Barbosa et al. (13) propuseram uma instancialização do Helios utilizando criptografia baseada em reticulados (Lattices) e provas de conhecimento zero pós-quânticas. A proposta substitui o esquema ElGamal por primitivas baseadas em problemas de reticulados (como LWE/SIS), visando garantir a segurança contra adversários quânticos, embora com custos computacionais e de tamanho de chaves significativamente maiores.

2.5 Infraestrutura de Mixnets (Verificatum)

Verificatum é uma biblioteca criptográfica robusta que implementa *mixnets* de recifração verificáveis, servindo como componente fundamental para diversos sistemas de votação eletrônica (14). Diferente de soluções completas como Helios ou ElectionGuard, o Verificatum foca especificamente na execução segura e auditável do embaralhamento de votos cifrados, utilizando provas de conhecimento zero (especialmente a prova de Terelius e Wikström (9)) para garantir que a saída é uma permutação válida da entrada, sem revelar a correspondência. A arquitetura multi-servidor oferece fortes garantias de anonimato: a

dissociação entre identidade e voto é preservada desde que pelo menos um servidor seja honesto.

Limitações: A operação requer infraestrutura distribuída robusta e coordenação entre múltiplos servidores. A implementação de referência, apesar de escrita principalmente em Java, contém chamadas a código nativo em C, sujeitando-a a vulnerabilidades de memória típicas dessa linguagem. Além disso, por utilizar re-cifração ElGamal, a privacidade é computacional, não perpétua. Outro fator limitante é a dependência da Java Virtual Machine (JVM), o que dificulta a integração direta com o software embarcado das urnas eletrônicas brasileiras (UE2020 e posteriores), que operam em ambiente restrito compatível com C/C++ e Rust.

2.6 O Protocolo E2Easy

O protocolo E2Easy (7) é uma proposta de votação presencial fim-a-fim cujo objetivo central é combinar verificabilidade pública forte com um fluxo de uso simples no posto de votação. O eleitor pode verificar que seu voto foi registrado como pretendido (através de desafios opcionais) e posteriormente confirmar que foi contabilizado corretamente no resultado final, tudo sem depender de infraestrutura especial (conectividade de rede, dispositivos auxiliares) ou de materiais distribuídos previamente (códigos impressos, cartões). Qualquer observador pode auditar a apuração a partir de um boletim público único contendo compromissos criptográficos e provas verificáveis.

2.7 Posicionamento da Proposta

Este trabalho apresenta o E2Easy-PC, uma adaptação do protocolo original que substitui as primitivas criptográficas baseadas em reticulados por Compromissos de Pedersen sobre curvas elípticas, mantendo a estrutura e fluxo operacional do protocolo. A motivação central dessa substituição é garantir privacidade perpétua através de compromissos com ocultação perfeita, enquanto a implementação em Rust elimina vulnerabilidades de memória. O Capítulo 5 detalha essas escolhas e suas implicações de segurança.

2.7.1 Análise Comparativa

Comparando o E2Easy-PC com as soluções revisadas, identificam-se diferenças arquiteturais e de segurança significativas:

Privacidade Eterna vs. Computacional: Sistemas baseados em criptografia homomórfica ElGamal — incluindo Helios (4), Belenios (5), ElectionGuard (3) e Verificatum (14) — oferecem ocultação computacional: o sigilo dos votos depende da intratabilidade do problema do logaritmo discreto. Caso essa suposição seja quebrada no futuro (e.g.,

por algoritmos quânticos de Shor), a privacidade de eleições passadas será comprometida retrospectivamente. Em contraste, o E2Easy-PC emprega Compromissos de Pedersen com ocultação perfeita (*Perfect Hiding*), garantindo que o sigilo do voto jamais possa ser quebrado, independentemente de avanços computacionais futuros, incluindo computadores quânticos. Esta propriedade, denominada *everlasting privacy* (privacidade eterna), é particularmente crítica em sistemas eleitorais, onde a confidencialidade do voto deve ser preservada perpetuamente.

Complexidade Operacional: ElectionGuard e CHVote (11) requerem infraestrutura complexa: cerimônias criptográficas distribuídas, distribuição prévia de materiais (códigos de verificação), múltiplos servidores coordenados. O E2Easy-PC adota uma arquitetura autocontida e minimalista, adequada para urnas eletrônicas isoladas (*air-gapped*) operando de forma autônoma, alinhando-se ao modelo brasileiro de votação presencial em DRE.

Essa simplicidade, contudo, introduz um *trade-off* importante: a premissa de um dispositivo confiável para a privacidade. No E2Easy-PC, a urna eletrônica detém sozinha as chaves de abertura dos votos (os nonces r) até o momento da apuração. Isso cria um ponto único de falha para a privacidade: se a urna for comprometida e os nonces vazados antes do embaralhamento, o sigilo dos votos daquela seção pode ser quebrado. Embora essa característica simplifique drasticamente a operação (eliminando a necessidade de múltiplos guardiões de chaves e servidores de mixagem online), ela exige confiança na segurança física e lógica do equipamento (*hardware security modules*, *air-gapping* rigoroso) para proteger os segredos temporários.

Segurança de Memória e Compatibilidade: Verificatum e ElectionGuard dependem majoritariamente de Java e C++, respectivamente, herdando vulnerabilidades de memória ou exigindo *runtimes* pesados. Embora exista uma implementação experimental do ElectionGuard em Rust (validando a tendência de migração para linguagens seguras), a versão de produção permanece em C++. Helios e Belenios utilizam Python e OCaml/JavaScript, adequados para cenários web, mas não otimizados para sistemas embarcados críticos. O E2Easy-PC, implementado integralmente em Rust, garante *memory safety* em tempo de compilação, eliminando classes inteiras de vulnerabilidades. Além disso, por compilar para código nativo sem dependência de *runtime* pesado ou *Garbage Collector*, o Rust oferece a eficiência e compatibilidade necessárias para o hardware restrito das urnas eletrônicas.

Modelo de Votação: Scantegrity (12) foca em cédulas de papel com tinta especial, modelo incompatível com DRE puramente eletrônico. CHVote depende de materiais distribuídos previamente pelo correio, inviável para o modelo presencial brasileiro. O E2Easy-PC foi concebido especificamente para o contexto de votação presencial com terminal eletrônico autocontido.

Tabela 1 – Comparação entre Soluções de Votação E2E

Solução	**Privacidade**	**Mixnet**	**Integração Urna**	**Segurança Memória**
Helios	Computacional	Não	Baixa (Web)	Parcial
ElectionGuard	Computacional	Não	Média	Parcial (C++)
Verificatum (Mixnet)	Computacional	Sim	Baixa (JVM)	Baixa (Java/C)
E2Easy-PC	**Incondicional**	**Sim**	**Alta (Nativa)**	**Alta (Rust)**

Em síntese, o E2Easy-PC posiciona-se como uma solução que combina verificabilidade universal (*mixnets* com provas ZKP), privacidade perpétua (Compromissos de Pedersen), segurança de software (Rust) e simplicidade operacional (sem cerimônias complexas ou materiais prévios). Projetado especificamente para urnas eletrônicas isoladas operando em DRE, o sistema mantém baixa latência no registro de votos e realiza a geração de provas apenas na fase final de apuração, sem impactar a experiência do eleitor. Esta combinação representa uma contribuição ao estado da arte em sistemas E2E para votação presencial.

3 Fundamentação Teórica

3.1 Verificabilidade Fim-a-Fim (E2E)

A verificabilidade fim-a-fim (E2E) requer que o processo eleitoral publique evidências criptográficas suficientes para que participantes independentes verifiquem, sem confiar em segredos operacionais, três critérios: (i) *cast-as-intended* (lançado como pretendido), (ii) *recorded-as-cast* (gravado como lançado), e (iii) *tallied-as-recorded* (apurado como gravado), *consistente* com o conjunto de votos publicados (3, 2). Na prática, (i) é implementado por desafios no ato da votação (e.g., *Benaloh challenge*), embora estudos recentes apontem desafios na eficácia da verificação humana e sugiram modelos de "verificação tutelada" para mitigar a coerção e erros do eleitor (15); (ii) por provas de correção do escrutínio (provas de embaralhamento, decifração verificada, verificação homomórfica); e (iii) pela publicação de boletins e artefatos auditáveis que permitem recomputar o resultado.

Formalmente, o sistema expõe um boletim público com registros criptográficos dos votos e as provas associadas, de modo que qualquer verificador execute algoritmos que aceitam com alta probabilidade somente quando as propriedades acima forem satisfeitas. Auditorias de risco (RLA) complementam a verificabilidade, oferecendo garantias estatísticas de que o resultado tabulado coincide com uma amostra física quando aplicável (10); em votação exclusivamente eletrônica com *mixnets* e provas, as RLAs são complementares e dependem do modelo operacional.

3.2 Notação

Seja $p \in \mathbb{Z}$ um número primo, e seja $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ o corpo finito de inteiros módulo p. Denote por $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ o subgrupo multiplicativo de $\mathbb{Z}_p$, e por $\mathbb{Z}_p^2 = \{y \in \mathbb{Z}_p^* \mid \exists x \in \mathbb{Z}_p^* : y \equiv x^2 \pmod{p}\}$ o grupo dos resíduos quadráticos em $\mathbb{Z}_p^*$. Assumimos parâmetros seguros com $p = 2q + 1$, onde q é primo, de modo que $\mathbb{Z}_p^2$ seja um subgrupo cíclico de ordem q. Denote por $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$ o conjunto de inteiros módulo q, usado para representar expoentes e escalares nas operações de grupo.

As operações dentro de $\mathbb{Z}_p^*$ e de seus subgrupos são realizadas módulo p: elementos do grupo são multiplicados entre si (e.g., $g^a \cdot g^b$) e exponenciados por escalares de $\mathbb{Z}_q$ (e.g., g^a para $a \in \mathbb{Z}_q$). Operações aritméticas sobre expoentes e escalares (elementos de $\mathbb{Z}_q$) — soma e multiplicação — são realizadas módulo q. A redução modular é omitida por concisão. Denotamos por $g \in \mathbb{Z}_p^2$ um gerador do subgrupo, isto é, um elemento cujo conjunto de potências g^a, com $a \in \mathbb{Z}_q$, gera todo $\mathbb{Z}_p^2$ (com $g \neq 1$). A amostragem uniforme é denotada por $x \leftarrow\!\$\, \mathcal{S}$, significando que x foi escolhido uniformemente de um conjunto ou

grupo finito $\mathcal{S}$; por exemplo, $a \leftarrow\$\ \mathbb{Z}_q$ indica um escalar escolhido uniformemente em $\mathbb{Z}_q$. Ressalta-se que, embora este capítulo utilize a notação multiplicativa para generalidade, a implementação descrita nos capítulos subsequentes adota a notação aditiva de curvas elípticas (detalhada na Seção 3.5).

3.3 Compromissos de Pedersen

Um esquema de compromisso criptográfico é uma primitiva que permite a um participante fixar um valor secreto m de modo que este seja mantido oculto até um momento posterior, quando o participante opte por revelá-lo, provando que o valor revelado corresponde exatamente ao originalmente fixado. Formalmente, um esquema de compromisso consiste em uma função $C = \mathsf{Com}(m, r)$ que gera um valor público C (o compromisso) a partir de uma mensagem m e de uma aleatoriedade r. Para verificar se um par (m, r) abre corretamente o compromisso C, basta recomputar $C' = \mathsf{Com}(m, r)$ e verificar se $C = C'$.

As propriedades de segurança fundamentais exigidas são a *hiding* (ocultação), que garante que C não revela informação computacionalmente útil sobre m, e a *binding* (vinculação), que assegura ser computacionalmente inviável exibir duas aberturas distintas $(m, r) \neq (m', r')$ para o mesmo compromisso C após sua publicação.

Sejam $g, h \in \mathbb{Z}_p^2$ geradores públicos e independentes (cujas relações exponenciais entre si são desconhecidas) de $\mathbb{Z}_p^2$. Para uma mensagem $m \in \mathbb{Z}_q$ e um valor aleatório $r \leftarrow\$\ \mathbb{Z}_q$, o compromisso de Pedersen é definido por

$$C \;=\; g^r\, h^m.$$

Dizemos que o par (m, r) constitui uma abertura válida para C quando a igualdade acima é satisfeita.

O esquema de Pedersen satisfaz *perfectly hiding* (ocultação perfeita): para qualquer mensagem $m \in \mathbb{Z}_q$, quando $r \leftarrow\$\ \mathbb{Z}_q$, a distribuição de C é estatisticamente indistinguível da distribuição uniforme sobre $\mathbb{Z}_p^2$, de modo que C não revela informação sobre m. Ademais, sob a hipótese de intratabilidade do problema do logaritmo discreto no subgrupo de ordem q, o esquema satisfaz *computationally binding* (vinculação computacional): é inviável encontrar dois pares distintos $(m, r) \neq (m', r')$ tais que $g^r h^m = g^{r'} h^{m'}$.

Uma propriedade fundamental do esquema de Pedersen é o homomorfismo aditivo. Dados dois compromissos $C_1 = g^{r_1} h^{m_1}$ e $C_2 = g^{r_2} h^{m_2}$, tem-se

$$C_1 \cdot C_2 \;=\; g^{r_1 + r_2} h^{m_1 + m_2},$$

ou seja, o produto de compromissos corresponde ao compromisso da soma das mensagens (com a soma das aleatoriedades). Essa propriedade permite operações sobre valores sob compromisso sem a necessidade de abertura. Em particular, dado um compromisso $C =$

$g^r h^m$ a uma mensagem m e escolhendo uma nova aleatoriedade $r' \leftarrow\$ \mathbb{Z}_q$, é possível gerar um recompromisso

$$C' \;=\; C \cdot g^{r'} \;=\; g^{r+r'} h^m,$$

que constitui um compromisso válido à mesma mensagem m, mas com aleatoriedade $r + r'$. Do ponto de vista de um observador externo que desconhece r e r', os compromissos C e C' são indistinguíveis, mesmo ocultando o mesmo valor m. Essa operação de recompromisso é central em protocolos de embaralhamento verificável, onde é necessário realeatorizar compromissos preservando seu conteúdo.

3.4 Provas de Embaralhamento Verificáveis

Apresenta-se, de forma resumida, o objeto provado em um *restricted shuffle* (embaralhamento verificável) e o esboço da prova de Terelius–Wikström (9), como empregada em plataformas como a Verificatum (14).

3.4.1 Enunciado (compromissos)

Seja $G = \langle g \rangle$ um grupo cíclico de ordem prima q, e sejam $C = (C_1, \ldots, C_n)$ e $D = (D_1, \ldots, D_n)$ listas públicas de compromissos de Pedersen sobre mensagens em $\mathbb{Z}_q$. Um embaralhamento (*shuffle*) correto atesta a existência de uma permutação ψ de $\{1, \ldots, n\}$ e de um vetor de aleatoriedades $\mathbf{r} = (r_1, \ldots, r_n) \in \mathbb{Z}_q^n$ tais que, para todo $i \in \{1, \ldots, n\}$,

$$D_i \;=\; C_{\psi(i)}\, g^{r_i}.$$

Isto é, D resulta de C por uma permutação acompanhada de recompromisso (realeatorização) elemento a elemento.

3.4.2 Esboço da prova (Terelius–Wikström)

A prova estabelece, sem revelar ψ nem $\mathbf{r}$, que D é um recompromisso permutado de C. O protocolo se desenrola em três etapas: (1) o provador publica compromissos auxiliares codificando a permutação e as realeatorizações; (2) um desafio público é gerado via hash da transcrição (Fiat–Shamir), transformando a prova de interativa para não interativa; (3) as checagens do verificador consistem em igualdades algébricas em G que vinculam os compromissos auxiliares às listas C e D, verificando que o mesmo multiconjunto de mensagens foi preservada por permutação e que cada D_i é um recompromisso válido. A solidez (*soundness*) decorre da hipótese de dificuldade do logaritmo discreto em G; a ocultação (zero-conhecimento) advém da ocultação perfeita dos compromissos e da simulabilidade do transcript gerado.

3.4.3 Propriedades e custos

O protocolo é completo, sólido sob DLog e zero-conhecimento. O tamanho da prova e o custo de verificação são lineares em n, com verificações paralelizáveis. A não-interatividade obtida via Fiat–Shamir permite que a prova seja publicada uma única vez num boletim público, sem exigir comunicação entre provador e verificador.

3.5 Curvas Elípticas

Os algoritmos e protocolos apresentados podem ser formulados de maneira equivalente sobre curvas elípticas (EC) definidas sobre corpos finitos. Para tanto, substitui-se a notação multiplicativa por notação aditiva: exponenciação g^a torna-se multiplicação escalar $a \cdot G$, e produtos $g^a \cdot g^b$ tornam-se somas de pontos $a \cdot G + b \cdot G$. Esta mudança é uma reescrita algébrica: as propriedades (associatividade, comutatividade, identidade) permanecem.

Uma curva elíptica E sobre $\mathbb{F}_p$ é o conjunto de pontos $(x, y) \in \mathbb{F}_p^2$ satisfazendo $y^2 = x^3 + ax + b$ (com condição de não-singularidade $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$), mais o ponto no infinito $\mathcal{O}$ como elemento identidade. Denota-se $E(\mathbb{F}_p)$ o conjunto de todos os pontos, formando um grupo abeliano sob adição geométrica de pontos. O número de pontos é $\#E(\mathbb{F}_p)$. Para fins criptográficos, seleciona-se $\#E(\mathbb{F}_p) = q \cdot h$ onde q é primo (ordem do subgrupo) e h é pequeno (cofator, tipicamente 1). Um ponto gerador $G \in E(\mathbb{F}_p)$ de ordem q gera um subgrupo cíclico de ordem q.

Na notação adotada neste trabalho, pontos são representados em maiúsculas (C, H, G) e escalares em minúsculas (m, r, a, b). Para o esquema de Pedersen em EC com geradores $G, H \in E(\mathbb{F}_p)$:

$$C = r \cdot G + m \cdot H,$$

onde $r, m \in \mathbb{Z}_q$, as operações são: multiplicações escalares $r \cdot G$ e $m \cdot H$, seguidas de adição de pontos. A realeatorização é $C' = C + r' \cdot G = (r + r') \cdot G + m \cdot H$, onde $(r + r')$ é calculado módulo q.

Curvas elípticas oferecem **segurança equivalente com chaves muito menores**: 256 bits em EC correspondem a aproximadamente 3072 bits em RSA. Neste trabalho, adotamos a curva **NIST P-256** (secp256r1), padronizada pelo NIST, com ordem $q \approx 2^{256}$ (primo) e cofator 1, oferecendo 128 bits de segurança clássica. A curva é amplamente implementada em bibliotecas auditadas e adotada em padrões modernos (TLS, ECDSA).

4 Protocolo E2Easy-PC

4.1 Visão Geral

Este capítulo descreve o protocolo E2Easy-PC, uma adaptação para curvas elípticas do protocolo E2Easy original (7) (detalhes das substituições criptográficas no Capítulo 5). O protocolo consiste em uma tupla de algoritmos

$$(\mathsf{Setup}, \mathsf{onStart}, \mathsf{onVoterActive}, \mathsf{onChallenge}, \mathsf{onFinish}, \mathsf{verifyVote}, \mathsf{validateFiles})$$

utilizados pela Autoridade Eleitoral, equipamentos de votação, mesários, eleitores e fiscais eleitorais, operando sobre estruturas de dados criptográficas publicadas como artefatos auditáveis.

4.2 Mecanismos Criptográficos

O protocolo utiliza os seguintes mecanismos criptográficos para garantir a segurança e auditabilidade do processo eleitoral:

4.2.1 Compromissos de Pedersen

Os votos são ocultados através de Compromissos de Pedersen em curvas elípticas (definição completa na Seção 3.3):

$$\mathsf{Com}(m; r) = r \cdot G + m \cdot H$$

onde G e H são geradores independentes da curva, m é a mensagem (voto) e r é a aleatoriedade. Este esquema oferece ocultação perfeita e vinculação computacional.

4.2.2 Códigos de Rastreio

Os Códigos de Rastreio H_i formam uma cadeia de hash que vincula todos os votos sequencialmente, conforme a fórmula $H_i = \mathsf{Hash}(H_{i-1}, T_i, \mathsf{com}_i)$, permitindo que cada eleitor verifique que seu voto foi registrado no RDCV sem revelar seu conteúdo.

4.2.3 Prova de Embaralhamento

O embaralhamento dos compromissos utiliza a Prova de Terelius-Wikström (9) adaptada para curvas elípticas (fundamentação teórica na Seção 3.4), que demonstra que com' é uma permutação recompromissada válida de com. Formalmente, sejam $C = (C_1, \ldots, C_n)$

e $D = (D_1, \ldots, D_n)$ listas de compromissos de Pedersen. A prova atesta que existe uma permutação ψ e aleatoriedades $\mathbf{r} = (r_1, \ldots, r_n)$ tais que:

$$D_i = C_{\psi(i)} + r_i \cdot G \quad \text{para todo } i \in \{1, \ldots, n\}$$

onde G é o gerador base da curva elíptica. A prova $\pi = (t, s, c, \hat{c})$ é não-interativa (via Fiat-Shamir), zero-conhecimento e computável em tempo linear em n, sem revelar ψ nem $\mathbf{r}$.

4.2.4 Assinaturas Digitais

Todas as estruturas de dados críticas (RDCV, RDCV', RDV', ZKPOutput) são assinadas digitalmente usando ECDSA sobre a curva NIST P-256 para garantir autenticidade e não-repúdio.

4.3 Estruturas de Dados

4.3.1 RDV - Registro Digital do Voto

O RDV (Registro Digital do Voto) é o registro tradicional do sistema eleitoral brasileiro, contendo votos em ordem lexicográfica. No contexto deste protocolo, não utilizamos diretamente o RDV tradicional.

4.3.2 RDV' - RDV Embaralhado

O RDV' (RDV Embaralhado) contém os votos $v'_{x,i}$ revelados após a abertura dos compromissos em com', com suas identidades desvinculadas através do embaralhamento, garantindo que a identidade dos eleitores não possa ser rastreada até seus votos.

O protocolo opera sobre as seguintes estruturas de dados, que são publicadas como artefatos auditáveis ao final da eleição, acompanhadas de provas criptográficas verificáveis.

4.3.3 RDCV - Registro Digital do Compromisso do Voto

O RDCV (Registro Digital do Compromisso do Voto) é uma estrutura fundamental que mantém uma *hash chain* (cadeia de hash) de todos os compromissos de votos durante a eleição. Cada entrada i contém:

- Código de Rastreio: $\mathcal{H}_i = \mathsf{Hash}(\mathcal{H}_{i-1}, T_i, \mathsf{com}_i)$
- Horário (*timestamp*): T_i
- Lista de compromissos de votos: $\mathsf{com}_i = (\mathsf{com}_{i,1}, \ldots, \mathsf{com}_{i,M})$

A integridade do RDCV é garantida através da cadeia de hash, iniciada pela cauda $\mathcal{H}_0 = \mathsf{Hash}(Q, \mathsf{PublicSignKey})$ e finalizada pela cabeça $\mathcal{H} = \mathsf{Hash}(\mathcal{H}_{\text{last}}, \text{“CLOSE”})$.

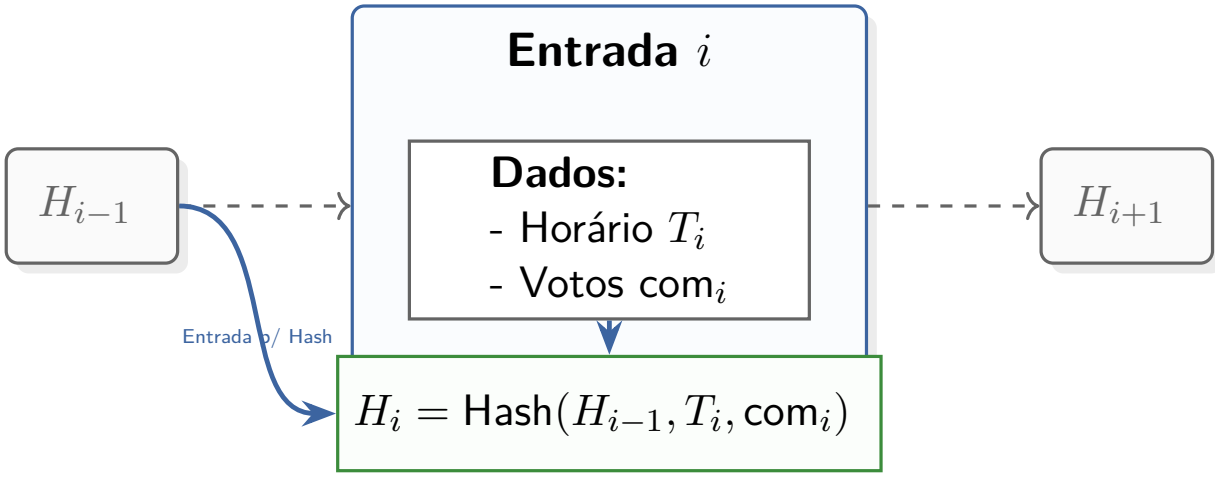

Figura 1 – Estrutura da Cadeia de Hash do RDCV.

4.3.4 RDCV' - RDCV Embaralhado

O RDCV' contém os mesmos compromissos do RDCV após aplicação de permutação ψ e recompromisso:

$$\mathsf{com}' = (\mathsf{com}'_0, \ldots, \mathsf{com}'_{n-1})$$

onde cada com'_i é obtido pela permutação e adição de novo nonce r'_i, garantindo o anonimato dos eleitores.

4.3.5 ZKPOutput - Saída da Prova Zero-Conhecimento

O ZKPOutput contém a prova zero-conhecimento de embaralhamento de Terelius-Wikström (9), que demonstra que com' é uma permutação recompromissada de com (detalhes teóricos na Seção 3.4). Esta estrutura inclui:

- Prova de embaralhamento $\pi = (t, s, c, \hat{c})$
- Lista de votos abertos embaralhados (correspondentes a com' no RDV')
- Lista de nonces utilizados na abertura

4.4 Fases do Protocolo

O protocolo executa as seguintes fases, operando sobre as estruturas descritas acima:

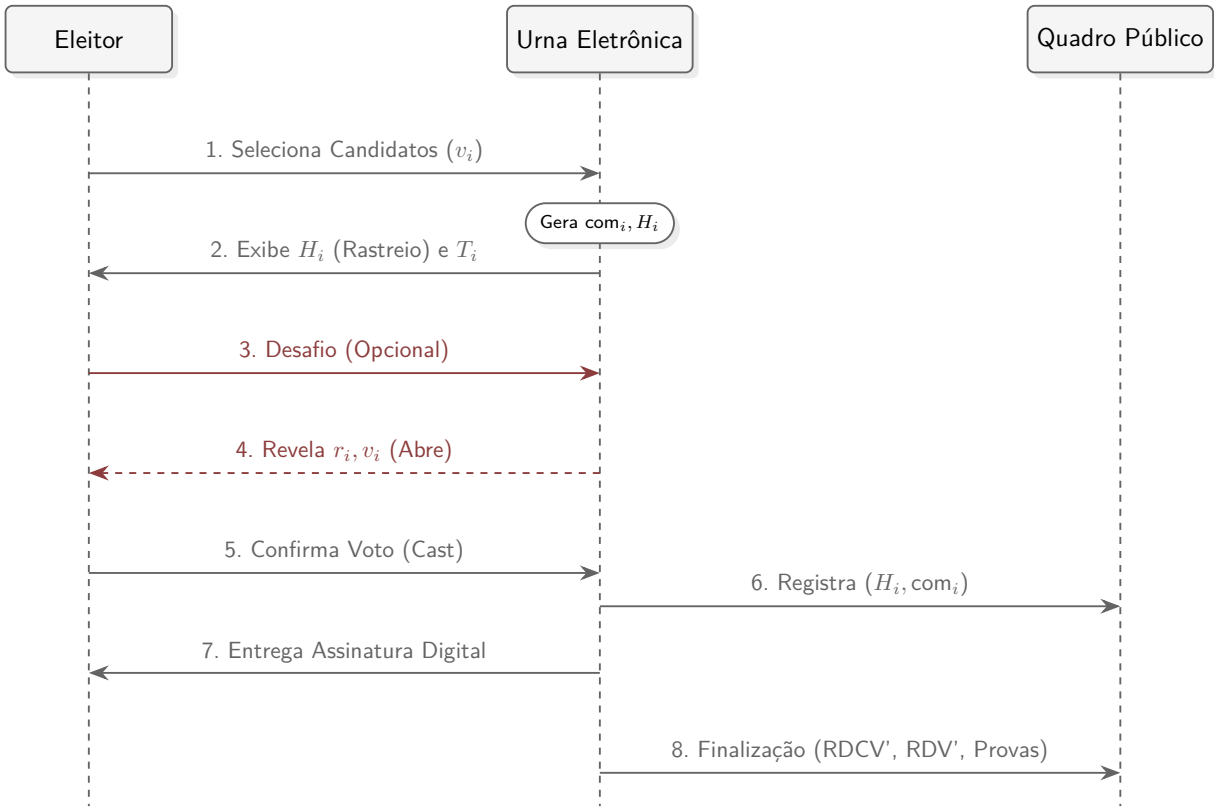

Figura 2 – Fluxo de interação entre Eleitor, Urna e Quadro Público.

4.4.1 Configuração Inicial (Setup(λ))

Estabelece os parâmetros criptográficos para um dado nível de segurança λ:

- Seleciona uma curva elíptica apropriada, define o gerador G
- Cria geradores independentes H_i para o Compromisso de Pedersen
- Cria o conjunto de dados infoContest relacionados às disputas da eleição (número de disputas, quantas escolhas são permitidas em cada uma)

4.4.2 Início da Eleição (onStart(A, infoContest))

Inicializa a Urna Eletrônica para operação:

- Gera par de chaves de assinatura (se necessário)
- Define Q como string de configuração especificando os algoritmos utilizados
- Define $\mathcal{H}_0 = \mathsf{Hash}(Q, \mathsf{PublicSignKey})$ como a cauda do Código de Rastreio
- Cria o arquivo RDCV — uma tabela vazia contendo: coluna de Código de Rastreio, coluna de horário, e uma coluna para cada um dos M cargos em disputa

- Publica Q e a chave pública de assinatura

4.4.3 Votação (onVoterActive($v_{i,1}, \ldots, v_{i,M}$))

Durante a fase de votação, para cada eleitor i que lança M votos (um para cada cargo em disputa), a urna:

- Define T_i como o horário (*timestamp*)
- Para cada voto $v_{i,j}$ com $j \in [1, M]$:
  - Define $r_{i,j} \xleftarrow{\$} \mathbb{Z}_q$
  - Cria o compromisso do voto: $\mathsf{com}_{i,j} \leftarrow \mathsf{Com}(v_{i,j}; r_{i,j})$
- Agrupa os compromissos: $\mathsf{com}_i \leftarrow (\mathsf{com}_{i,1}, \ldots, \mathsf{com}_{i,M})$
- Cria o Código de Rastreio: $\mathcal{H}_i = \mathsf{Hash}(\mathcal{H}_{i-1}, T_i, \mathsf{com}_i)$
- Fornece $\mathcal{H}_i$ e T_i ao eleitor
- Mantém em memória: com_i, $r_i = (r_{i,1}, \ldots, r_{i,M})$, e $v_i = (v_{i,1}, \ldots, v_{i,M})$

4.4.4 Desafio ou Lançamento (onChallenge($Cast$))

Após gerar um voto, o eleitor pode optar por duas ações distintas:

- **Cast = True** (Lançamento): O eleitor confirma seu voto. A urna:
  - Assina digitalmente o Código de Rastreio $\mathcal{H}_i$
  - Grava $\mathcal{H}_i$, T_i e com_i no arquivo RDCV
  - Fornece a assinatura digital ao eleitor
- **Cast = False** (Desafio): O eleitor desafia a urna. A urna:
  - Fornece $\mathcal{H}_{i-1}$, com_i e r_i ao eleitor para verificação
  - Descarta $\mathcal{H}_i$, T_i, com_i, v_i e r_i da memória

4.4.5 Finalização (onFinish())

A fase de finalização é a última função executada pela urna e compreende o processo de embaralhamento e decifração dos votos. Ela:

- Define $\mathcal{H} = \mathsf{Hash}(\mathcal{H}_{\text{last}}, \text{“CLOSE”})$ como cabeça do Código de Rastreio e grava no RDCV
- A partir do RDCV, cria lista com todos os compromissos: $\mathsf{com} \leftarrow (\mathsf{com}_0, \ldots, \mathsf{com}_n)$

- Escolhe função de permutação ψ e nonces $r'_{i,j}$ para cada compromisso
- Embaralha e realeatoriza os compromissos da lista com, gerando com′
- Calcula abertura de cada compromisso em com′ usando r_i, v_i (em memória), ψ e $r'_{i,j}$
- Cria os arquivos:
  - RDCV': contém os compromissos embaralhados com′
  - RDV': contém todos os votos abertos embaralhados $v'_{x,i}$
  - ZKPOutput: contém prova zero-conhecimento de embaralhamento e aberturas dos compromissos
- Assina digitalmente todos os arquivos criados
- Salva no disco: RDV', RDCV, RDCV', ZKPOutput e as assinaturas

4.5 Apuração dos Resultados

A apuração no protocolo E2Easy-PC é um processo determinístico e publicamente verificável que ocorre imediatamente após a fase de finalização. Uma vez que a função onFinish produziu o arquivo **RDV'** (Registro Digital do Voto Embaralhado), que contém a lista de votos decifrados e desvinculados de seus autores originais, a totalização dos resultados consiste na contagem simples das escolhas registradas neste arquivo.

Diferentemente de sistemas convencionais onde a totalização ocorre em memória ou em banco de dados protegido, aqui a "urna" entrega os votos em claro (mas anonimizados) para o público. Qualquer observador de posse do arquivo RDV' pode realizar a contagem independente e chegar ao mesmo resultado final. A validade desse resultado depende estritamente da verificação de que o conteúdo de RDV' corresponde a uma permutação válida dos votos cifrados originais (garantido pela prova de embaralhamento) e que os votos cifrados correspondem às intenções dos eleitores (garantido pela verificação individual e desafios).

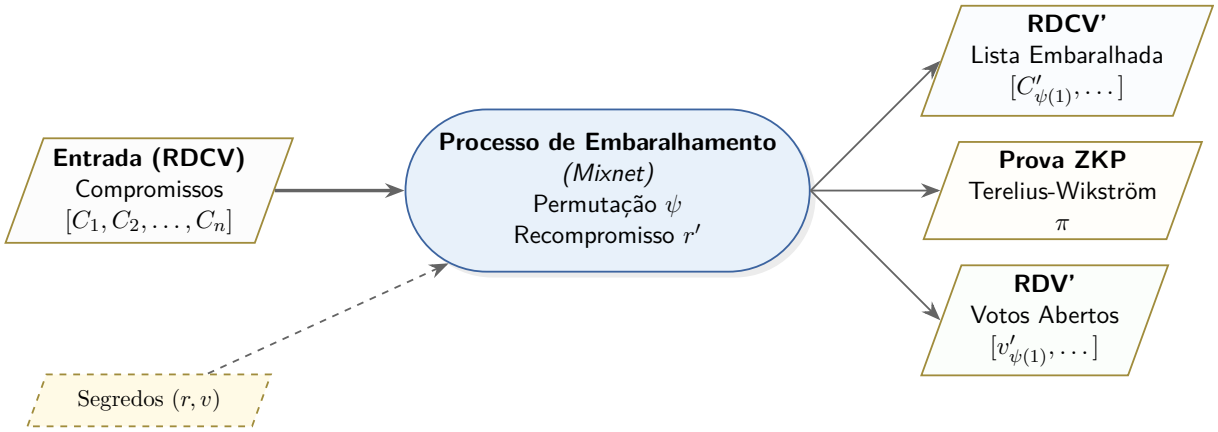

Figura 3 – Fluxo de dados da Mixnet e geração de provas.

4.6 Verificação

O protocolo E2Easy suporta dois tipos de verificação:

4.6.1 Verificação Individual (verifyVote$(\mathcal{H}_i, \mathcal{H}_{i-1}, T_i, v_i, r_i)$)

Permite que cada eleitor (ou mesário) verifique que um voto desafiado está correto. O verificador recalcula os compromissos $\widehat{\mathsf{com}}_j \leftarrow \mathsf{Com}(v_{i,j}; r_{i,j})$ para cada voto $v_{i,j}$ com $j \in [1, M]$, concatena-os em $\widehat{\mathsf{com}}$, calcula $\widehat{\mathcal{H}}_i = \mathsf{Hash}(\mathcal{H}_{i-1}, T_i, \widehat{\mathsf{com}})$ e verifica se $\widehat{\mathcal{H}}_i = \mathcal{H}_i$.

4.6.2 Verificação Universal (validateFiles())

Permite que qualquer observador (fiscais eleitorais, eleitores, auditores) verifique a integridade completa da eleição:

- Verifica assinaturas digitais dos arquivos RDV', RDCV, RDCV' e ZKPOutput
- Verifica cadeia de hash do RDCV usando $\mathcal{H}_0$, com_i e T_i
- Verifica que a prova de embaralhamento π em ZKPOutput está correta e corresponde aos compromissos em RDCV e RDCV'
- Verifica que a abertura dos compromissos em ZKPOutput está correta e corresponde aos compromissos em RDCV' e votos às claras embaralhados em RDV'
- Retorna verdadeiro se todas as verificações são válidas, falso caso contrário

5 Adaptações do Protocolo e Análise de Segurança

Este capítulo detalha as divergências fundamentais entre o protocolo E2Easy original (7) e a variante E2Easy-PC implementada neste trabalho. Discutimos as motivações para a substituição das primitivas criptográficas baseadas em reticulados por curvas elípticas, analisando o compromisso estratégico entre a integridade pós-quântica e a privacidade eterna (*everlasting privacy*), bem como a escolha da linguagem Rust sob a ótica da segurança de software.

5.1 Substituição de Primitivas Criptográficas

O protocolo E2Easy original foi concebido com foco em segurança pós-quântica completa (PQC), utilizando primitivas baseadas em reticulados (*Module-LWE* e *Module-SIS*) para os compromissos BDLOP e assinaturas Dilithium. Embora essa abordagem ofereça resistência contra adversários quânticos futuros tanto para integridade quanto para privacidade, ela acarreta complexidade de implementação e custos de desempenho significativos, além de depender de suposições de dureza criptográfica relativamente recentes em comparação com a criptografia clássica.

Nesta implementação, optamos por uma abordagem transicional (16), substituindo as primitivas de reticulados por equivalentes clássicos maduros e eficientes:

- **Compromissos**: Substituição do esquema BDLOP por Compromissos de Pedersen sobre a curva NIST P-256.
- **Assinaturas**: Substituição do Dilithium por ECDSA sobre a curva NIST P-256.
- **Prova de Embaralhamento**: Substituição da prova de embaralhamento baseada em reticulados utilizada no protocolo original (uma construção teórica baseada na dureza de problemas de reticulados como SIS) pelo protocolo de Terelius-Wikström (9). Esta mudança é significativa: enquanto a prova original buscava eficiência em estruturas de reticulados complexas, a prova de Terelius-Wikström é o padrão-ouro para *mixnets* baseadas em Logaritmo Discreto, oferecendo uma estrutura de verificação mais simples, madura e amplamente auditada na literatura (utilizada, por exemplo, no sistema Verificatum).

Essa substituição simplifica a implementação e permite o uso de bibliotecas auditadas e otimizadas, focando o esforço de pesquisa na arquitetura de segurança do software e na eficiência da *mixnet* paralela.

5.2 Privacidade Eterna vs. Integridade Pós-Quântica

A principal contribuição teórica desta adaptação reside na análise da privacidade a longo prazo. Em sistemas de votação, a privacidade do eleitor deve ser preservada perpetuamente, enquanto a integridade da apuração precisa ser garantida apenas durante o período da eleição e auditoria.

O esquema de compromisso BDLOP, na parametrização adotada pelo protocolo original, possui a propriedade de ocultação computacional (*computational hiding*), baseada na dureza do problema LWE. Isso implica que, se um computador quântico capaz de resolver LWE for construído no futuro, ele poderia, em tese, quebrar o sigilo dos votos passados. É importante ressaltar que reticulados *podem* ser configurados para oferecer ocultação estatística (*statistical hiding*), garantindo privacidade eterna, mas tal configuração exigiria parâmetros significativamente maiores, aumentando o tamanho das provas em aproximadamente 5 vezes, o que impactaria o desempenho em dispositivos restritos.

Em contraste, os Compromissos de Pedersen (17) utilizados nesta variante oferecem ocultação perfeita (*perfectly hiding*) com chaves muito menores. Isso significa que o valor do compromisso $C = r \cdot G + m \cdot H$ é estatisticamente independente da mensagem m. Mesmo um adversário com poder computacional infinito (incluindo computadores quânticos) não pode extrair m de C sem conhecer r, pois para qualquer mensagem candidata m' existe um r' válido que satisfaz a equação.

Portanto, ao sacrificar a integridade pós-quântica (que depende da vinculação computacional, quebrável por Shor), ganhamos *everlasting privacy*. Assumindo que computadores quânticos relevantes não estarão disponíveis durante a curta janela de tempo da eleição para forjar votos (quebra de integridade), o sistema garante que o sigilo dos eleitores permanecerá inviolável para sempre, uma propriedade superior à oferecida por esquemas baseados puramente em suposições computacionais de reticulados.

Figura 4 – Matriz de Trade-off: Integridade vs. Privacidade.

5.3 Segurança de Memória e a Escolha do Rust

A segurança do processo eleitoral não depende apenas de algoritmos criptográficos, mas da correção de sua implementação. Relatórios recentes de agências de segurança, como a NSA (8), indicam que a grande maioria das vulnerabilidades exploráveis em software decorre de erros de gerenciamento de memória (e.g., *buffer overflows*, *use-after-free*), comuns em linguagens como C e C++.

O protocolo original E2Easy possui uma implementação de referência em C. Nossa implementação em Rust elimina, por design, essas classes de vulnerabilidades através de seu sistema de tipos e *borrow checker* (verificador de empréstimo), garantindo *memory safety* em tempo de compilação sem o custo de um *garbage collector*. Esta escolha alinha a infraestrutura de votação com as melhores práticas modernas de desenvolvimento de software crítico, assegurando que a complexidade do protocolo não introduza brechas de segurança na aplicação.

6 Implementação em Rust

Este capítulo descreve os detalhes técnicos da implementação do protocolo E2Easy-PC. O sistema foi desenvolvido com o objetivo de garantir segurança de memória, alto desempenho e clareza de código, servindo como uma implementação de referência para votação presencial verificável baseada em *mixnets*.

6.1 Visão Geral e Ferramentas

A linguagem escolhida para o desenvolvimento foi **Rust**, conforme justificado no Capítulo 5. O ecossistema do Rust, através do gerenciador de pacotes *Cargo*, facilita a gestão de dependências e a compilação cruzada, enquanto as garantias de *memory safety* em tempo de compilação eliminam classes inteiras de vulnerabilidades.

Foram utilizadas *crates* (bibliotecas) consolidadas da comunidade Rust para funções críticas:

- **`p256`**: Implementação pura em Rust das curvas elípticas NIST P-256, utilizada para todas as operações de grupo (pontos e escalares).
- **`sha2`**: Para funções de hash criptográfico (SHA-256), essenciais para a cadeia de integridade (RDCV) e para o protocolo Fiat-Shamir nas provas de conhecimento zero.
- **`serde`** e **`serde_json`**: Framework de serialização e desserialização, utilizado para estruturar os artefatos da eleição (votos, provas, configurações) em formato JSON legível e interoperável.
- **`rayon`**: Biblioteca de paralelismo de dados, fundamental para acelerar as operações matemáticas intensivas durante a fase de embaralhamento e geração de provas.

6.2 Arquitetura do Sistema

O software foi estruturado de forma modular, separando as primitivas criptográficas da lógica de negócio eleitoral. A arquitetura segue o padrão de uma *core library* (biblioteca central) que expõe as funcionalidades para executáveis de linha de comando (CLI). Os principais módulos são:

- **`types`**: Define as estruturas de dados fundamentais (Voto, RDCV, Prova de Embaralhamento) e suas regras de serialização.

- **`pedersen`**: Implementa o esquema de compromissos de Pedersen sobre a curva P-256.
- **`shuffler`**: Contém a lógica de *mixnet*, incluindo a permutação, re-randomização e a geração da prova de embaralhamento verificável.
- **`e2easy`**: O módulo principal que implementa a máquina de estados do protocolo (Votar, Desafiar, Depositar, Apurar).
- **`verifier`**: Implementa a lógica de verificação das provas criptográficas, utilizada pelas ferramentas de auditoria.

6.3 Estruturas de Dados

A persistência e a transparência do processo dependem de estruturas de dados bem definidas. O formato JSON foi adotado para todos os arquivos públicos, facilitando a auditoria por ferramentas independentes. As estruturas abaixo refletem diretamente os tipos definidos no módulo `types.rs`.

6.3.1 RDV'

O RDV' (`RDVPrime`) contém o resultado final da eleição: os votos em claro, porém desvinculados dos eleitores. Sua estrutura é um vetor de objetos `Vote`, onde cada voto é composto pelos campos `contest` (identificador do cargo) e `choice` (número do candidato escolhido).

6.3.2 RDCV

O RDCV é a estrutura central de auditoria, implementada como uma lista encadeada criptograficamente. Em Rust, a estrutura `RDCV` encapsula:

- `tail`: O código de rastreio inicial ($\mathcal{H}_0$), que ancora a cadeia de hash.
- `entries`: Um vetor de `CommittedBallot`, onde cada cédula contém:
  - `tracking_code`: O hash atual da cadeia (H_i).
  - `committed_votes`: Vetor de pontos na curva elíptica (`Element`) representando os compromissos de votos.
  - `timestamp`: Carimbo de tempo do registro.
- `head`: O código de rastreio final ($\mathcal{H}_{\text{last}}$), assinado digitalmente ao encerrar a eleição.

6.3.3 RDCV'

O RDCV' (`RDCVPrime`) representa a saída da mixnet antes da abertura. Ele contém apenas um campo `entries`, que é um vetor de pontos na curva elíptica (`Vec<Element>`). Estes pontos correspondem aos votos originais após passarem pela permutação ψ e re-randomização (re-compromisso), desvinculando-os de seus autores originais.

6.3.4 ZKPOutput - Prova de Embaralhamento

A estrutura `ZKPOutput` consolida todas as informações necessárias para a verificação pública do embaralhamento. Ela armazena:

- `shuffle_proof`: A prova de Terelius-Wikström propriamente dita, contendo os vetores de compromissos intermediários (t) e as respostas aos desafios escalares (s).
- `m_list` e `r_list`: Vetores contendo as aberturas dos compromissos (mensagem m e aleatoriedade r) do RDCV', permitindo verificar que correspondem aos votos em claro no RDV'.

6.4 Serialização Canônica e Consistência Criptográfica

A integridade de um sistema de votação depende da capacidade de reproduzir exatamente os mesmos hashes para os mesmos dados em diferentes máquinas. Em Rust, estruturas como `HashMap` não garantem ordem de iteração determinística, e implementações padrão de JSON podem variar a ordem das chaves ou a formatação de espaços em branco.

Para garantir que assinaturas digitais e cadeias de hash sejam verificáveis de forma consistente, implementamos uma estratégia de **serialização canônica**:

- **Ordenação de Campos**: Estruturas complexas são convertidas para vetores de bytes seguindo uma ordem estrita e pré-definida de seus campos.
- **Listas e Vetores**: Elementos de vetores (como a lista de compromissos no RDCV) são processados sequencialmente, garantindo que a ordem posicional influencie o hash resultante.
- **Tipos Primitivos**: Inteiros e pontos da curva elíptica são serializados em formatos binários fixos (e.g., *Little Endian* para inteiros, formato comprimido SEC1 para pontos), eliminando ambiguidades de representação textual.

6.4.1 Coordenadas Projetivas vs. Afins

Um aspecto crucial da implementação é a distinção entre a representação interna e externa dos pontos da curva elíptica. Internamente, a biblioteca `p256` utiliza **coordenadas projetivas** (X, Y, Z) para realizar as operações aritméticas (somas e multiplicações escalares), evitando a custosa operação de inversão modular a cada passo.

No entanto, coordenadas projetivas não possuem representação única (o ponto (X, Y, Z) é equivalente a $(\lambda X, \lambda Y, \lambda Z)$ para qualquer $\lambda \neq 0$). Para fins de serialização e *hashing*, os pontos são obrigatoriamente convertidos para **coordenadas afins** (x, y) e codificados no formato comprimido SEC1 (33 bytes). Essa conversão garante uma representação canônica única para cada ponto, essencial para a consistência das assinaturas e hashes, eliminando a ambiguidade introduzida pelo fator de escala Z.

Vale ressaltar que essa distinção técnica não compromete a legibilidade do código. A biblioteca `p256` implementa a sobrecarga de operadores matemáticos (traits `Add`, `Mul`), permitindo operações mistas transparentes entre pontos em coordenadas afins e projetivas. Dessa forma, a implementação permanece fiel à notação matemática abstrata (e.g., $C = r \cdot G + m \cdot H$), abstraindo a complexidade da representação interna dos operandos.

Essa abordagem assegura que o cálculo $H = \mathsf{Hash}(\text{Objeto})$ seja sempre idêntico, independentemente da arquitetura do processador ou da versão do compilador, requisito fundamental para a validação das assinaturas digitais e da integridade do RDCV.

6.5 Primitivas Criptográficas

O sistema opera sobre o grupo da curva elíptica NIST P-256, onde pontos representam compromissos e escalares representam votos e aleatoriedades. A implementação do esquema de Pedersen (17) segue a definição teórica apresentada no Capítulo 3, garantindo ocultação perfeita e vinculação computacional.

6.6 Lógica do Protocolo

A lógica central do protocolo é gerenciada pela estrutura `E2Easy`, que implementa a máquina de estados descrita no Capítulo 4 (Votação, Desafio/Lançamento, Finalização). O sistema mantém o estado da eleição (chaves, RDCV, geradores) e gerencia a criação de `TempBallot` (cédulas temporárias) durante a fase de votação, a persistência no RDCV durante o lançamento, e a invocação do módulo `Shuffler` durante a apuração.

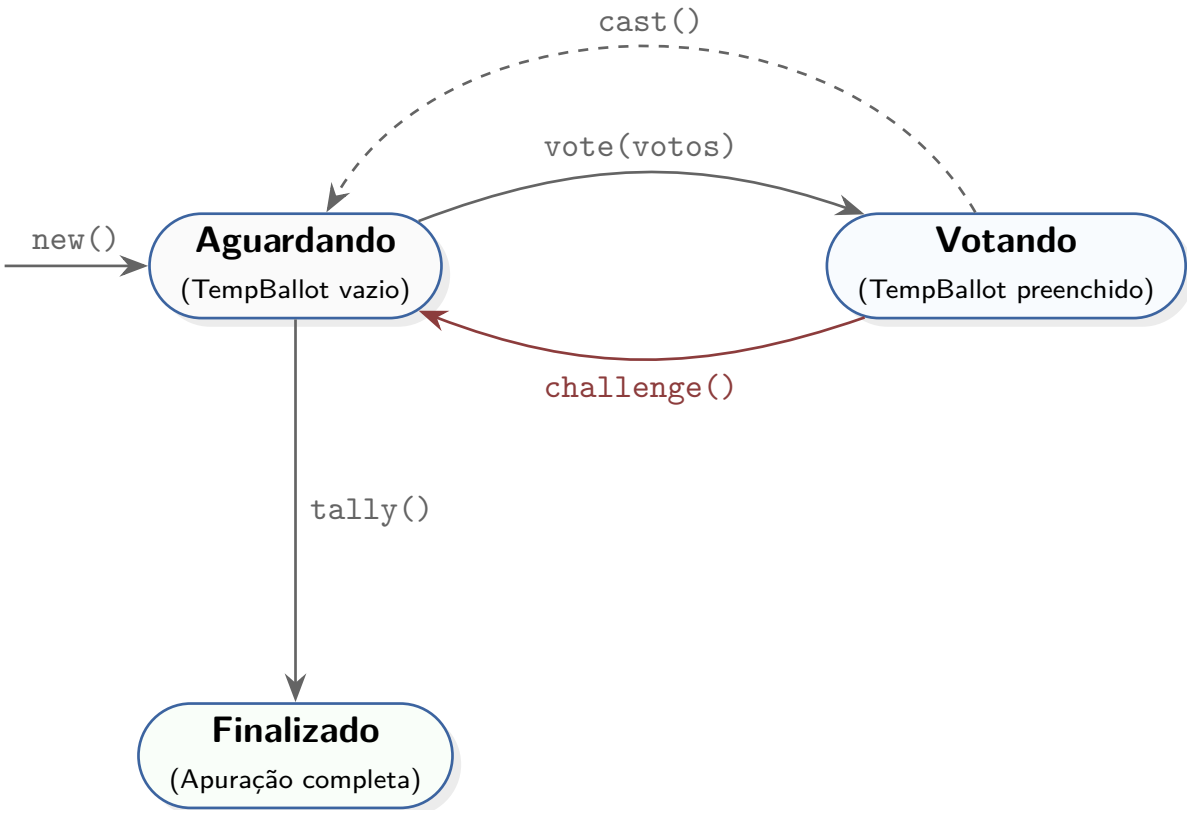

Figura 5 – Máquina de Estados da estrutura `E2Easy`.

6.7 Mixnet e Provas de Embaralhamento

O módulo `Shuffler` implementa o protocolo de embaralhamento descrito nas Seções 3.4 e 4.4.3. O processo combina permutação ψ, re-randomização dos compromissos e geração da prova de Terelius-Wikström (9). A implementação garante que nenhum voto seja alterado, inserido ou removido durante o embaralhamento.

6.7.1 Paralelismo e Desempenho

A geração da prova de embaralhamento envolve operações matriciais pesadas e exponenciações múltiplas. Para tornar o sistema viável em eleições de maior porte, a implementação utiliza a biblioteca `rayon` para paralelizar os cálculos. Especificamente, o cálculo dos vetores de desafio e as operações sobre os vetores de compromissos são distribuídos entre os núcleos da CPU.

O sistema foi dimensionado e testado considerando um parâmetro $N = 3000$ votos, que representa uma estimativa de pior caso para o número de eleitores em grandes seções eleitorais brasileiras. Graças ao paralelismo, o tempo de geração da prova e de verificação permanece dentro de limites operacionais aceitáveis para a apuração na própria seção eleitoral.

6.8 Legibilidade e Auditabilidade

A auditabilidade de um sistema de votação não se resume à verificação das provas criptográficas (auditabilidade de dados), mas estende-se à capacidade de inspeção do código-fonte (auditabilidade de software). A escolha do Rust e a estruturação do código priorizaram a clareza, permitindo uma correspondência direta entre a especificação teórica e a implementação.

6.8.1 Correspondência com a Especificação Matemática

No módulo `Shuffler`, a implementação dos algoritmos de embaralhamento e geração de provas segue rigorosamente os pseudocódigos propostos por Haenni et al. (18). A expressividade do Rust permite que operações vetoriais e aritméticas sobre curvas elípticas sejam escritas de forma quase idêntica à notação matemática, reduzindo a lacuna semântica que frequentemente esconde erros em implementações em C/C++.

6.8.2 Clareza no Fluxo de Controle

No módulo principal `E2Easy`, a lógica do protocolo é centralizada em uma estrutura unificada que gerencia os artefatos da eleição. O fluxo de operações (Votar, Desafiar, Depositar) é implementado através de métodos explícitos que manipulam o estado interno da urna de forma sequencial e previsível. Essa organização permite que auditores rastreiem o ciclo de vida do voto — da geração do compromisso até sua persistência no RDCV — validando a lógica de negócio diretamente pela leitura do código, sem a complexidade de gerenciamento manual de memória típica de implementações em C.

6.9 Ferramentas de Verificação e CLI

Para operacionalizar e auditar o protocolo, foram desenvolvidas ferramentas de linha de comando (*Command Line Interface* - CLI) que consomem a biblioteca principal:

- **`main`**: Executável de simulação que realiza o ciclo completo da eleição (configuração, geração de N votos, embaralhamento e verificação), utilizado para *benchmarks* e validação do fluxo.
- **`verificador_universal`**: Ferramenta autônoma destinada a auditores. Ela lê os arquivos públicos da eleição (RDCV, RDCV', RDV' e a Prova de Embaralhamento) e executa o algoritmo de verificação da prova ZKP. Se a verificação for bem-sucedida, garante-se matematicamente que o resultado (RDV') corresponde exatamente aos votos depositados (RDCV) após embaralhamento válido.

- **`verificador_individual`**: Ferramenta para validar desafios individuais. Permite que o eleitor (ou mesário) verifique que os compromissos e o *tracking code* revelados durante um desafio correspondem aos votos e nonces fornecidos pela urna, conforme descrito em `onChallenge`.

Além dos binários em Rust, o repositório inclui scripts de análise em Python (na pasta `Analysis`), utilizados para processar os logs de desempenho e gerar gráficos que relacionam o tempo de execução com o número de votos N.

7 Resultados

Neste capítulo, apresentamos a avaliação de desempenho da implementação do protocolo E2Easy-PC desenvolvida em Rust. O objetivo central da análise experimental é quantificar o impacto da substituição das primitivas baseadas em reticulados (Lattices) por curvas elípticas (ECC) e verificar os ganhos de eficiência obtidos através do paralelismo de dados proporcionado pela biblioteca `rayon`.

7.1 Metodologia e Ambiente de Testes

Os experimentos compararam o desempenho da implementação original do E2Easy (escrita em C e baseada na biblioteca FLINT para aritmética de reticulados) com a proposta E2Easy-PC (escrita em Rust e baseada na *crate* `p256`). Para garantir uma comparação justa, os testes foram normalizados considerando o número de eleitores N, onde cada eleitor realiza escolhas para $M = 6$ cargos distintos (parâmetro fixo em ambas as implementações para simular uma eleição complexa).

Os benchmarks foram executados em uma máquina com as seguintes especificações:

- **Processador**: Intel Core i5-10300H (4 cores/8 threads), frequência base de 2,50 GHz
- **Memória RAM**: 16 GB
- **Sistema Operacional**: Debian GNU/Linux 12 (bookworm)
- **Compiladores**: g++ 12.2.0 para o E2Easy original, Rust 1.91.1 (`rustc`/`cargo`) para o E2Easy-PC
- **Verificatum**: Versão 3.1.0, com otimização JVM (`-Xms2g -Xmx4g -XX:+UseParallelGC`)

A métrica principal utilizada foi o número de **ciclos de CPU**, capturados através da instrução `rdtsc` (ou equivalente de alta precisão), o que permite uma avaliação independente da frequência de operação momentânea do processador. Onde pertinente, apresentamos estimativas de tempo considerando uma frequência base de referência de 3.0 GHz (10^9 ciclos $\approx$ 0.33 segundos). Para o Verificatum, o tempo de computação da fase de embaralhamento foi extraído do próprio sistema (*Computation time*) e convertido para ciclos utilizando a mesma frequência de referência.

Os testes variaram o número de eleitores N de 100 a 500, com incrementos de 100. Cada medição representa a média de 10 execuções independentes para mitigar ruídos do sistema operacional.

7.2 Desempenho da Urna Eletrônica

A fase de interação com o eleitor compreende duas operações críticas: a geração do voto (`onVoterActive`) e o registro ou desafio do voto (`onChallenge`).

7.2.1 Fase de Votação

A Figura 6 apresenta o custo computacional para a geração dos compromissos e códigos de rastreio. Observa-se uma diferença de ordem de magnitude entre as abordagens. Enquanto a implementação original baseada em Lattices consome cerca de 5×10^7 ciclos por eleitor ($\approx$ 17 ms), o E2Easy-PC reduz esse custo para a faixa de 5×10^6 ciclos ($\approx$ 1,7 ms). Essa eficiência superior das operações em curvas elípticas (multiplicação escalar em P-256 vs operações matriciais em reticulados) garante uma experiência de uso instantânea, eliminando qualquer latência perceptível durante a seleção dos candidatos.

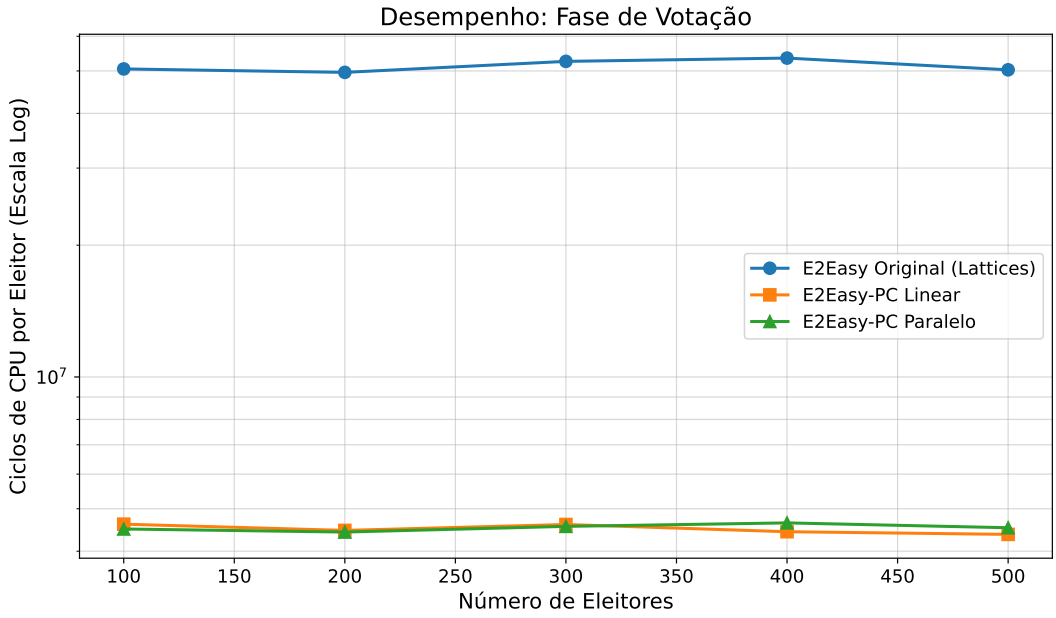

Figura 6 – Comparação de desempenho na fase de votação (Escala Logarítmica).

7.2.2 Fase de Lançamento

Na fase de lançamento (Figura 7), onde ocorre a assinatura digital e a persistência do voto, a implementação original baseada em Lattices apresentou um custo na faixa de 1,2 a 1,8 $\times 10^6$ ciclos por eleitor, enquanto a implementação em Rust reduziu-se para aproximadamente 4,8 $\times 10^5$ ciclos. Novamente, observa-se o ganho de eficiência das operações em curvas elípticas. Em termos absolutos, ambos os valores representam menos de 1 milissegundo em uma CPU moderna, um tempo irrelevante para a operação da urna.

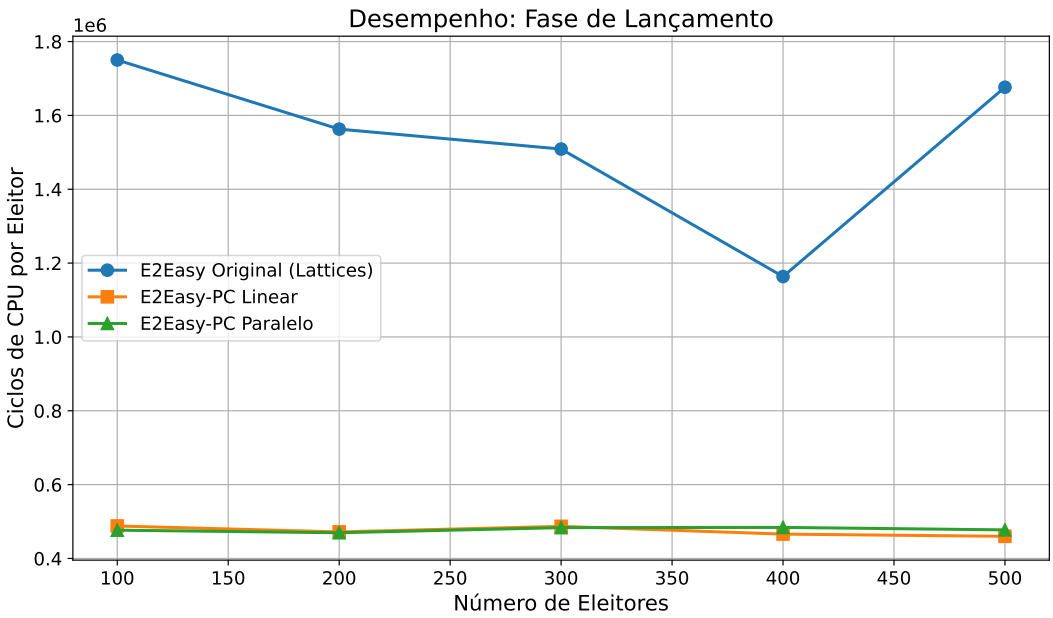

Figura 7 – Comparação de desempenho na fase de lançamento.

7.3 Desempenho da Apuração (Mixnet)

A fase de embaralhamento e geração da Prova de Conhecimento Zero (ZKP) é a etapa mais intensiva do protocolo, executada ao final da eleição (`onFinish`).

7.3.1 Comparação Geral

A Figura 8 ilustra o custo total do embaralhamento. A implementação original (Lattices) apresenta um custo na ordem de 5×10^{11} ciclos para 500 eleitores ($\approx$ 2 min 46 s). A transição para o E2Easy-PC (Compromissos de Pedersen) reduziu drasticamente esse custo, mesmo na execução sequencial (Linear), para a ordem de 4×10^{10} ciclos ($\approx$ 13,3 s). O uso de curvas elípticas, portanto, oferece um ganho de desempenho significativo para a geração das provas de Terelius-Wikström.

Para contextualizar esses resultados frente ao estado da arte, incluímos na análise o Verificatum 3.1.0, um sistema de mix-net de referência em Java, testado sob idênticas condições (P-256, `width=1`). Para pequenas cargas ($N = 100$), o E2Easy-PC Paralelo mostra-se 9$\times$ mais eficiente (2.0×10^9 ciclos ou $\approx$ 0,67 s vs 1.8×10^{10} ciclos ou $\approx$ 6,0 s do Verificatum). Contudo, conforme N cresce, essa vantagem diminui: em $N = 500$ eleitores, o E2Easy-PC consome 1.7×10^{10} ciclos ($\approx$ 5,67 s) contra 4.4×10^{10} do Verificatum ($\approx$ 14,67 s), uma razão de apenas 2.6$\times$. Essa convergência sugere que o Verificatum seria eventualmente mais eficiente para um número muito grande de entradas. No entanto, tal cenário não é o fator determinante nesta aplicação, visto que cada urna eletrônica processa e embaralha localmente um número relativamente pequeno de votos (tipicamente até 500

eleitores por seção), faixa onde o E2Easy-PC mantém vantagem de desempenho. Ambas as implementações demonstram viabilidade prática para cenários reais de votação.

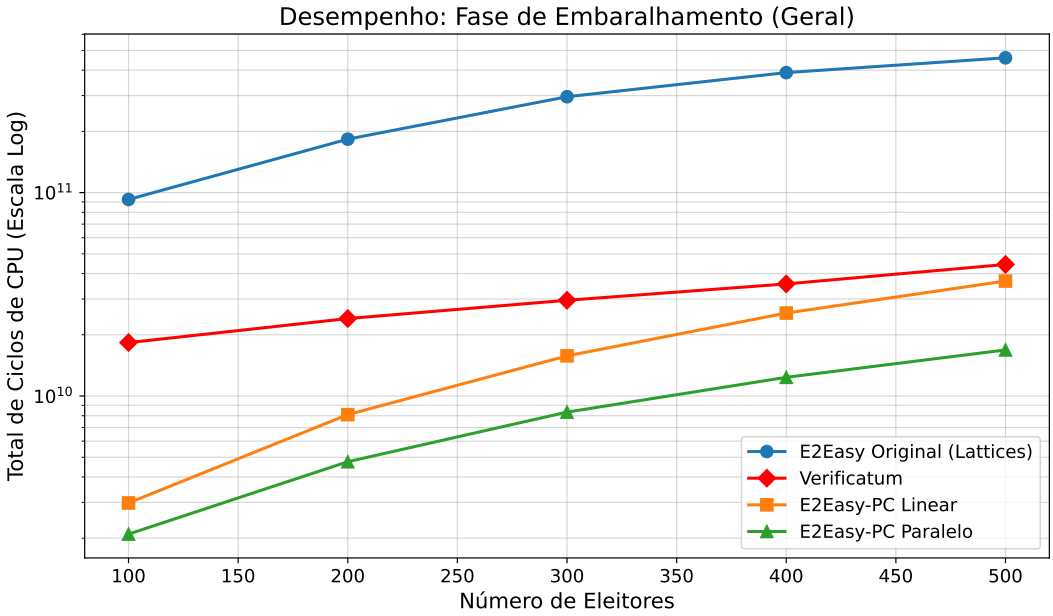

Figura 8 – Custo total da fase de embaralhamento (Escala Logarítmica).

7.3.2 Impacto do Paralelismo

Para avaliar o impacto da paralelização implementada com `rayon`, isolamos a comparação entre as versões Linear e Paralela do E2Easy-PC na Figura 9.

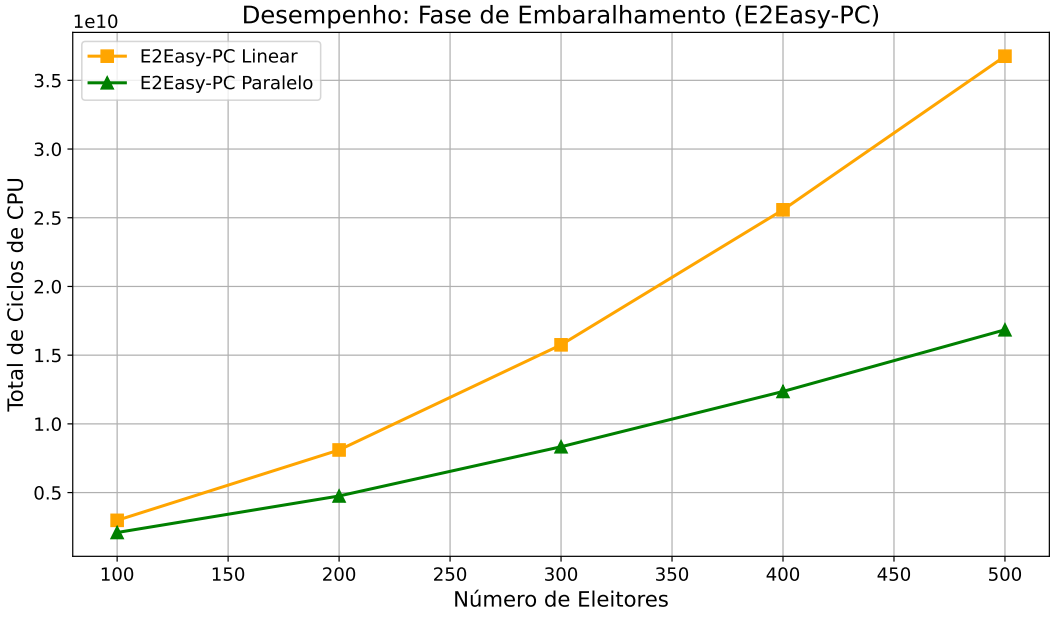

Figura 9 – Impacto do paralelismo na fase de embaralhamento do E2Easy-PC.

Os resultados demonstram um *speedup* consistente. Para $N = 500$ eleitores, a versão sequencial consumiu aproximadamente 3.7×10^{10} ciclos, enquanto a versão paralela reduziu para 1.7×10^{10} ciclos, um ganho de desempenho de aproximadamente $2.17\times$. Esse resultado confirma que a geração dos vetores de desafio e as operações de exponenciação modular, gargalos do algoritmo de Terelius-Wikström, são altamente paralelizáveis.

7.4 Escalabilidade

A análise dos gráficos indica um crescimento linear do tempo de execução em função do número de eleitores N, consistente com a complexidade $O(N)$ teórica da prova de embaralhamento.

7.5 Conclusão dos Resultados

Os experimentos validam a viabilidade prática do E2Easy-PC. A substituição de Lattices por Compromissos de Pedersen resultou em ganhos de desempenho em todas as fases. A implementação em Rust, além de oferecer garantias de segurança de memória, permitiu explorar o paralelismo de forma segura e eficaz, reduzindo o tempo de apuração pela metade em máquinas multi-core. O sistema demonstra escalabilidade linear, apto a suportar seções eleitorais de grande porte com tempos de resposta imediatos para o eleitor e apuração rápida para o mesário.

8 Conclusão

O desenvolvimento de sistemas de votação eletrônica auditáveis exige a conciliação de requisitos de segurança aparentemente conflitantes. O presente trabalho buscou resolver o dilema entre a integridade pós-quântica e a privacidade de longo prazo através do E2Easy-PC, uma adaptação do protocolo E2Easy para o contexto de curvas elípticas e implementação em linguagem segura.

8.1 Síntese e Análise de Resultados

A principal contribuição deste estudo foi a análise do *trade-off* de segurança adotado. Ao substituirmos os compromissos baseados em reticulados (Lattices) do protocolo original (7) por Compromissos de Pedersen em curvas elípticas, abdicamos da propriedade de integridade incondicional contra adversários quânticos em favor da *Everlasting Privacy* (Privacidade Eterna) (6).

Argumentamos que esta é a escolha estratégica correta para o cenário eleitoral. A integridade da apuração é uma propriedade que deve ser garantida durante a janela temporal da eleição e auditoria. Se assumirmos que computadores quânticos capazes de quebrar o Logaritmo Discreto não estarão operacionais *neste curto período*, a integridade do sistema permanece robusta. Por outro lado, o sigilo do voto é uma propriedade que deve perdurar indefinidamente. A propriedade de *Perfect Hiding* dos Compromissos de Pedersen assegura matematicamente que, mesmo com poder computacional infinito no futuro, é impossível violar a privacidade do eleitor, uma garantia que esquemas de ocultação computacional (como BDLOP ou ElGamal) não podem oferecer.

No aspecto de engenharia de software, a implementação em Rust demonstrou ser uma alternativa superior às implementações tradicionais em C/C++ e Java (como Verificatum (14)). A eliminação de vulnerabilidades de memória em tempo de compilação alinha o sistema às recomendações de agências de segurança de alto nível (8), reduzindo drasticamente a superfície de ataque da urna eletrônica.

Os resultados experimentais confirmaram a viabilidade prática da solução. As operações em curvas elípticas (NIST P-256) mostraram-se ordens de magnitude mais eficientes que a aritmética de reticulados para a fase de votação, garantindo uma experiência fluida ao eleitor. Na fase de apuração, o uso de paralelismo de dados com a biblioteca `rayon` permitiu gerar provas de embaralhamento para milhares de votos em tempo hábil, demonstrando que a segurança criptográfica forte não inviabiliza a operação em larga escala.

8.2 Limitações

A principal limitação teórica do E2Easy-PC reside na dependência da dureza do Logaritmo Discreto para a integridade. Embora improvável no curto prazo, a existência de um adversário quântico ativo durante a eleição permitiria a forja de provas de embaralhamento e assinaturas, comprometendo o resultado (embora sem revelar os votos).

Do ponto de vista prático, a implementação atual constitui um núcleo criptográfico (*backend*) e uma interface de linha de comando. A integração com hardware de votação real, interfaces gráficas para eleitores e mesários, e a gestão de chaves em ambiente hostil não foram escopo deste trabalho.

8.3 Trabalhos Futuros

A evolução natural deste projeto aponta para três direções principais:

1. **Reticulados com Ocultação Perfeita**: Para fechar a lacuna de integridade sem sacrificar a privacidade eterna, o próximo passo teórico é investigar o uso de esquemas de compromisso baseados em reticulados configurados para *Statistical Hiding*. Diferente da configuração padrão (*computational hiding*), essa abordagem exigiria parâmetros de reticulado significativamente maiores (dimensão e módulo) para garantir a uniformidade da distribuição dos compromissos.

2. **Aceleração por Hardware**: A adoção de reticulados com ocultação perfeita implicaria em um custo computacional e de armazenamento oneroso para CPUs convencionais (estimado em um aumento de 5x no tamanho das provas, e.g., de 150MB para 750MB). Embora viável, esse aumento torna o processamento "desagradável" em hardware de consumo. O desenvolvimento de aceleradores em hardware (FPGA ou ASIC) dedicados à aritmética de reticulados seria essencial para tornar essa "solução definitiva" eficiente em urnas eletrônicas.

3. **Verificação Formal**: Para elevar o nível de garantia da implementação em Rust, sugere-se a aplicação de métodos formais para provar matematicamente que o código corresponde à especificação do protocolo. Iniciativas como o projeto Hacspec e trabalhos recentes de análise formal de protocolos de votação em blockchain (19), bem como implementações verificadas como a *Open Vote Network* (20), oferecem o caminho para certificar que o software está livre não apenas de erros de memória, mas também de falhas lógicas na implementação das primitivas criptográficas.

Em conclusão, o E2Easy-PC representa um passo sólido em direção a sistemas de votação que respeitam tanto a soberania do voto popular quanto o direito inalienável do

cidadão ao sigilo, utilizando o estado da arte da engenharia de software para proteger a democracia na era digital.

Referências

1 TRIBUNAL SUPERIOR ELEITORAL. *TPS 2012: saiba tudo o que aconteceu na segunda edição do Teste Público de Segurança*. 2021. <https://www.tse.jus.br/comunicacao/noticias/2021/Setembro/tps-2012-saiba-tudo-o-que-aconteceu-na-segunda-edicao-do-teste-publico-de-seguranca>. Acesso em 13 set. 2021.

2 U.S. ELECTION ASSISTANCE COMMISSION. *End to End (E2E) Protocol Evaluation Process*. 2023. <https://www.eac.gov/voting-equipment/end-end-e2e-protocol-evaluation-process>. Acesso em 2 abr. 2023.

3 MICROSOFT. *ElectionGuard Glossary*. 2023. <https://www.electionguard.vote/overview/Glossary/>. Acesso em 2 abr. 2023.

4 ADIDA, B. Helios: Web-based open-audit voting. In: *USENIX Security Symposium*. San Jose, CA: USENIX Association, 2008. v. 17, p. 335–348.

5 CORTIER, V.; GAUDRY, P.; GLONDU, S. Belenios: a simple private and verifiable electronic voting system. In: *Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows*. Cham: Springer, 2019. (Lecture Notes in Computer Science, v. 11565), p. 214–238.

6 ARAPINIS, M. et al. Practical everlasting privacy. In: *Principles of Security and Trust*. Berlin: Springer, 2013. (Lecture Notes in Computer Science, v. 7796), p. 21–40.

7 COMINETTI, E. L. et al. E2Easy: a simple Lattice-based in-person end-to-end voting scheme. In: *Financial Cryptography and Data Security. FC 2025 International Workshops*. Cham: Springer, 2025. p. 281–296.

8 CYBERSECURITY AND INFRASTRUCTURE SECURITY AGENCY (CISA); NATIONAL SECURITY AGENCY (NSA). *The Case for Memory Safe Roadmaps*. Washington, DC, 2023. Acesso em 29 nov. 2024. Disponível em: <https://www.cisa.gov/resources-tools/resources/case-memory-safe-roadmaps>.

9 TERELIUS, B.; WIKSTRÖM, D. Proofs of restricted shuffles. In: *Progress in Cryptology – AFRICACRYPT 2010*. Berlin: Springer, 2010. (Lecture Notes in Computer Science, v. 6055), p. 100–113.

10 STARK, P. B. Super-simple simultaneous single-ballot risk-limiting audits. In: *2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 10)*. USENIX Association, 2010. Disponível em: <https://www.usenix.org/legacy/event/evtwote10/tech/full_papers/Stark.pdf>.

11 HAENNI, R. et al. *CHVote Protocol Specification*. Bern, Switzerland, 2017. Sistema desenvolvido para o cantão de Genebra e outros cantões suíços. Disponível em: <https://eprint.iacr.org/2017/325>.

12 CHAUM, D. et al. Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In: *Proceedings of the 2008 Electronic Voting Technology Workshop (EVT 08)*. San Jose, CA: USENIX Association, 2008.

13 BARBOSA, J. P. C. et al. From ElGamal to Lattice-based: Enhancing Helios voting with quantum-safe cryptography. In: *Applied Cryptography and Network Security (ACNS 2025)*. Cham: Springer, 2025. (Lecture Notes in Computer Science, v. 15767), p. 29–49.

14 WIKSTRÖM, D. *Verificatum Mix-Net*. 2024. <https://www.verificatum.org/>. Acesso em 21 jun. 2025.

15 ARAUJO, R. et al. Is Benaloh challenge suitable for polling station elections? In: *Ninth International Joint Conference on Electronic Voting (E-Vote-ID 2024)*. Bonn: Gesellschaft für Informatik, 2024. p. 61–65.

16 STEBILA, D.; MOSCA, M. Post-quantum key exchange for the internet and the open quantum safe project. In: *Selected Areas in Cryptography – SAC 2016*. Cham: Springer, 2017. (Lecture Notes in Computer Science, v. 10532), p. 14–37.

17 PEDERSEN, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In: *Advances in Cryptology — CRYPTO '91*. Berlin: Springer, 1992. (Lecture Notes in Computer Science, v. 576), p. 129–140.

18 HAENNI, R. et al. Pseudo-code algorithms for verifiable re-encryption mix-nets. In: *Financial Cryptography and Data Security*. Cham: Springer, 2017. (Lecture Notes in Computer Science, v. 10323), p. 370–387.

19 HANSEN, L. L.; NIELSEN, E. H.; SPITTERS, B. A formal security analysis of blockchain voting. In: *CoqPL 2024*. London, UK: ACM SIGPLAN, 2024. Workshop on Coq for Programming Languages. Disponível em: <https://raw.githubusercontent.com/hacspec/hacspec.github.io/refs/heads/master/coqpl24-paper8-2.pdf>.

20 HACSPEC CONTRIBUTORS. *Open Vote Network Specification in Hacspec*. 2024. <https://github.com/hacspec/specs/pull/16>. Implementação experimental. Acesso em 22 jun. 2025.