Department of Informatics

Privacy and Verifiability in Decentralized Electronic Voting Systems

Dissertation submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Christian Killer
from
Gebenstorf, Switzerland

approved in July 2023

at the request of
Prof. Dr. Burkhard Stiller
Prof. Dr. Salil Kanhere

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zürich, July 19, 2023

The Chairperson of the Doctoral Board: Prof. Elaine Huang, Ph.D.

© 2023 - *Christian Killer*
All rights reserved.

Abstract

Fair and secure voting is a cornerstone of democracies and, at the same time, one of the most challenging government processes. Trustworthy elections are crucial to ensure long-term stability and foster trust in the government. Historically, technology generally supported elections to ensure the correct organization and execution of electoral processes. The omnipresent digitization and emergence of the internet changed the fabric of societies and governments, naturally including voting systems. Thus, government services are increasingly digitized, and an essential effort is digitizing democratic processes, such as deploying Electronic Voting (EV) systems. A key difference and crucial aspect of voting processes is whether votes are cast in person or remotely (*e.g.*, via the internet), *i.e.*, Remote EV (REV) systems allow voters to remotely cast their ballot from an uncontrolled environment.

Digitizing voting processes involve complex cryptographic mechanisms because voting systems present a unique dichotomy between privacy and verifiability. The perfect balance of these properties depends heavily on the country's governmental structures (*e.g.*, electoral system, state of digitization) and, crucially, the public's acceptance and trust in the governmental entities. In EV systems, a Public Bulletin Board (PBB) serves as a tamper-proof audit trail, *i.e.*, as evidence of the correct execution of the voting protocol. Thus, the immutability of PBBs is a technical requirement to protect stored audit trails from tampering. Instead of relying on the honesty of governmental authorities and election officials, decentralized procedures can help mitigate centralized failures, sabotage risks, or the collusion of dishonest actors. Another fundamental challenge of publicly releasing the data stored on PBBs is ensuring that the data published, such as ballots, proofs, and logs, do not leak any privacy-sensitive data and does not impose any risks on voters. Once published, powerful adversaries can store information indefinitely and attempt to reveal the plain text of these data in the future.

Transparency is critical for voting systems and especially crucial in achieving verifiability properties. Thus, Blockchains (BC) and Distributed Ledgers (DL) emerged prominently as an instantiation

of the PBB in voting systems' architectures. A DL network with a *permissioned* consensus is suitable for voting systems, because it directly maps the trust model to a permissioned network architecture formed by trusted governmental authorities. Based on permissioned DLs, this thesis proposes novel identity management approaches for the Swiss Remote Postal Voting (RPV) system, as well as an End-to-End Verifiable (E2E-V) RPV system. Besides these systems strengthening the Swiss RPV, three REV systems are introduced, a fully decentralized voting system based on Homomorphic Encryption (HE) that achieves Receipt-Freeness and E2E-V, and fourth, its successors based on Re-Encryption Mixnets. Finally, a decentralized voting system achieving Unconditional Privacy is introduced, *i.e.*, its privacy guarantees do not depend on any computational hardness assumptions.

This thesis contains novel RPV and REV systems, which are evaluated quantitatively and qualitatively. Evaluations performed for systems contained within this thesis indicate that *(a)* permissioned DLs are suitable PBBs for cryptographic voting schemes, and *(b)* performance evaluations demonstrate that these systems scale up to Swiss electorate sizes. Finally, *(c)* all of these systems contain explicit assumptions in terms of their respective evaluations of security and trust boundaries, indicating novel threat events that emerge and their respective mitigations.

Kurzfassung

Faire und sichere Wahlen sind ein Eckpfeiler von Demokratien und gleichzeitig eine der größten Herausforderungen für Regierungen. Vertrauenswürdige Wahlen sind entscheidend, um langfristige Stabilität zu gewährleisten und das Vertrauen in die Regierung zu fördern. In der Vergangenheit wurden Wahlen in der Regel durch Technologie unterstützt, um die korrekte Organisation und Durchführung von Wahlprozessen zu gewährleisten. Die allgegenwärtige Digitalisierung haben das Gefüge von Gesellschaften und Regierungen und Abstimmungssysteme verändert. Staatliche Dienstleistungen werden zunehmend digitalisiert. Ein wesentliches Bestreben ist die Digitalisierung demokratischer Prozesse, wie z.B. der Einsatz von Electronic Voting (EV) Systemen. Ein entscheidender Aspekt von Wahlprozessen ist, ob die Stimmen persönlich vor Ort, oder aus der Ferne (z.B. über das Internet) abgegeben werden. D.h., Remote EV (REV)-Systeme ermöglichen es den WählerInnen, ihre Stimme aus der Ferne in einer unkontrollierten Umgebung abzugeben.

Digitalisierte Wahlprozesse erfordern komplexe kryptografische Mechanismen, da Wahlsysteme eine einzigartige Dichotomie zwischen Privatsphäre und Verifizierbarkeit darstellen. Das perfekte Gleichgewicht zwischen diesen Eigenschaften hängt stark von den staatlichen Strukturen des Landes (Wahlsystem, Stand der Digitalisierung) und vor allem von der Akzeptanz und dem Vertrauen der Öffentlichkeit in die staatlichen Stellen ab. In EV-Systemen dient ein Public Bulletin Board (PBB) als fälschungssicherer Prüfpfad, d.h., als Nachweis für die korrekte Ausführung des Wahlprotokolls. Daher ist die Unveränderlichkeit von PBBs eine technische Voraussetzung, um vor Manipulationen zu schützen. Anstatt sich auf die Integrität von Regierungsbehörden und Wahlhelfern zu verlassen, können dezentrale Verfahren dazu beitragen, zentralisierte Fehler, Sabotagerisiken oder das geheime Zusammenspiel unehrlicher Akteure abzuschwächen. Eine weitere grundlegende Herausforderung bei der Veröffentlichung von Daten, die auf PBBs gespeichert sind, besteht darin, sicherzustellen, dass die veröffentlichten Daten, wie z. B. Stimmzettel, Nachweise und Protokolle, keine datenschutzrechtlich sensiblen Daten preisgeben und keine Risiken für die WählerInnen mit sich bringen. Einmal veröf-

fentlicht, können mächtige Angreifer Informationen auf unbestimmte Zeit speichern und versuchen, den Klartext dieser Daten in der Zukunft offenzulegen.

Transparenz ist für Wahlsysteme von zentraler Bedeutung um Überprüfbarkeitseigenschaften zu erreichen. Daher haben sich Blockchains (BC) und Distributed Ledgers (DL) als eine Variante des Public Bulletin Board (PBB) in den Architekturen von Wahlsystemen durchgesetzt. Auch wenn der genehmigungsfreie Konsens für viele Anwendungsfälle (z. B. Finanztransaktionen oder autonome Organisationen) vorteilhaft ist, passen solche nicht zu Wahlsystemen, die bei demokratischen Wahlen eingesetzt werden, bei denen zuvor ein Mindestmaß an zuverlässigen und vertrauenswürdigen Behörden festgelegt wird. Ein verteiltes Netzwerk mit genehmigungsbasiertem Konsens ist für Wahlsysteme geeignet, da es das Vertrauensmodell direkt auf eine genehmigte Netzwerkarchitektur abbildet, die von vertrauenswürdigen Regierungsbehörden gebildet wird. Basierend auf genehmigungsbasierten DLs schlägt diese Arbeit einen neuartigen Ansatz von Identitätsmanagement für das briefliche Abstimmunssystem (RPV) vor, sowie ein neuartiges Ende-zu-Ende verifizierbares (E2E-V) briefliches Abstimmungssystem. Neben diesen Systemen, die auf das briefliches Abstimmungssystem der Schweizer abzielen, werden drei REV-Systeme vorgestellt: ein vollständig dezentrales Wahlsystem, das auf homomorpher Verschlüsselung (HE) basiert und Receipt-Freeness und E2E-V erreicht, und Weiterentwicklungen, die auf Re-Encryption Mixnets basieren. Schließlich wird ein dezentrales Wahlsystem vorgestellt, das Unconditional Privacy erreicht, d.h. dessen Datenschutzgarantien nicht von einem theoretischen Zahlenproglem abhängt, welches keine einfache Lösung hat.

Die vorliegende Arbeit enthält neuartige RPV- und REV-Systeme, die quantitativ und qualitativ bewertet wurden. Die durchgeführten Evaluationen zeigen, dass *(a)* genehmigungsbasierte DLs geeignete PBBs für kryptografische Abstimmungssysteme sind, und *(b)* Leistungsevaluationen zeigen, dass die Systeme bis zu Schweizer Wählergrößen skalieren. Schließlich enthalten alle Systeme explizite Annahmen, und die Systeme wurden im Hinblick auf ihre Sicherheits- und Vertrauensgrenzen evaluiert, wobei neuartige Bedrohungslagen und ihre jeweiligen Methoden zur Risikominderung aufgezeigt werden.

Acknowledgments

I grow little of the food I eat.
I do not make any of my own clothing.
I speak a language I did not invent or refine.
I did not discover the mathematics I use.
I am protected by freedoms and laws I did not conceive of and do not enforce.
I am moved by music I did not create.
I did not invent the transistor, the microprocessor, object-oriented programming, or most of the technology I work with.
I love and admire my species, living and dead.

Email from Steve Jobs to himself, (September 2, 2010 at 11:08 PM)

-

Thanks to my supervisor Prof. Dr. Burkhard Stiller, who helped me in all endeavors and allowed me to grow along the path of this PhD thesis.
Thanks to my co-supervisor Prof. Dr. Salil Kanhere for his valuable input.
Thanks to all the colleagues at the CSG, Dr. Muriel Franco, Dr. Eder Scheid, Dr. Bruno Rodrigues, Jan von der Assen, Dr. Alberto Huertas, and Dr. Thomas Bocek.
Thanks to the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement No. 830927, the CONCORDIA project.
Thanks to all the brilliant students that I had the pleasure to work with and supervise.
Thanks to my mother Hendrika. Thanks to Thomas and Hana.
Thanks to Roman, SMO. Thanks to Mao.
Thanks to all friends that supported me.
Thanks to my love, Melissa. When I first saw you, I knew.

Table of Contents

1 Introduction **2**

1.1 Remote Voting Systems . . . 4
1.2 Problem Statement . . . 6
1.3 Research Questions . . . 8
1.4 Thesis Outline . . . 9

2 Cryptographic Foundations **11**

2.1 Cryptographic Fundamentals . . . 12
2.2 Cryptographic Security . . . 13
2.2.1 Security Goals and Attack Models . . . 13
2.2.2 Hard Computational Problems . . . 14
2.3 Public-Key Cryptography . . . 16
2.3.1 ElGamal Public-Key Cryptosystem . . . 17
2.3.2 Digital Signatures . . . 19
2.3.3 Blind Signatures . . . 19
2.4 Commitment Schemes . . . 20
2.5 Secret Sharing . . . 21
2.5.1 Distributed Key Generation . . . 22
2.5.2 Cooperative Decryption . . . 23
2.5.3 Operational Concerns . . . 24
2.6 Homomorphic Encryption . . . 24
2.7 Re-Encryption . . . 25
2.8 Mixnets . . . 26
2.8.1 Decryption Mixnets . . . 26
2.8.2 Re-Encryption Mixnets . . . 27

2.9 Zero-Knowledge Proofs . . . 29
2.9.1 Non-Interactive ( ) . . . 30
2.9.2 Schnorr Identification Algorithm . . . 30
2.9.3 Chaum-Pedersen Protocol . . . 32
2.9.4 Designated Verifier Proof . . . 36
2.9.5 Shuffle Proof . . . 37

3 Voting System Fundamentals **40**
3.1 Remote Electronic Voting Requirements . . . 40
3.1.1 Protocol Requirements . . . 41
3.1.2 General Voting System Requirements . . . 47
3.2 From Centralized to Decentralized REV Systems . . . 48
3.2.1 Centralized REV System . . . 48
3.2.2 Decentralized REV System . . . 50
3.3 Requirements of BC-based E-Voting . . . 52
3.3.1 Trust and Reputation in Voting Systems . . . 52
3.3.2 Voting Scheme . . . 54
3.3.3 Voting System Design . . . 54
3.3.4 Voting System Implementation . . . 55
3.3.5 PBB Architecture and Deployment . . . 55
3.4 Overview of REV Approaches . . . 57
3.4.1 Encrypt and Sign Approach . . . 57
3.4.2 The Two Agencies Approach . . . 58
3.4.3 Homomorphic Encryption Approach . . . 59
3.4.4 Mix Network-based Approach . . . 61

4 Literature Review **62**
4.1 Review of Remote Electronic Voting Systems . . . 63
4.1.1 Multi-Authority protocol CGS97 . . . 64
4.1.2 Decentralized and Distributed . . . 66
4.1.3 Coercion-Resistance of JCJ05 . . . 68
4.1.4 Self-Tallying Scheme HRZ08 . . . 70
4.1.5 Commercial and Country-Wide Deployments . . . 71
4.1.6 Key Observations . . . 73
4.2 Review of Remote Postal Voting Schemes . . . 74

4.2.1 Verifiable Postal Voting . . . 75
4.2.2 Verifiable Remote Postal Voting with Paper Assurance . . . 76
4.2.3 STROBE-Voting . . . 77
4.2.4 RemoteVote . . . 78
4.2.5 SAFE Vote . . . 79
4.2.6 Key Observations . . . 80

5 Swiss Remote Postal Voting Process **82**
5.1 Legal Background . . . 83
5.2 Postal Voting Process Flow (PVPF) in Switzerland . . . 84
5.2.1 Setup Phase *A* . . . 85
5.2.2 Delivery Phase *B* . . . 88
5.2.3 Casting Phase *C* . . . 88
5.2.4 Storage Phase *D* . . . 88
5.2.5 Tallying Phase *E* . . . 89
5.2.6 Validation Phase *F* . . . 90
5.2.7 Destruction Phase *G* . . . 90
5.3 PVPF Risk Assessment (RA) . . . 90
5.3.1 RA of Setup Phase *A* . . . 91
5.3.2 RA of Delivery Phase *B* . . . 93
5.3.3 RA of Casting Phase *C* . . . 94
5.3.4 RA of Storage Phase *D* . . . 94
5.3.5 RA of Tallying Phase *E* . . . 94
5.3.6 RA of Validation Phase *F* and Destruction Phase *G* . . . 95
5.4 Conclusions on the PVPF . . . 96

6 Decentralized Remote Postal Voting **98**
6.1 Proverum . . . 99
6.1.1 System Design . . . 100
6.1.2 Implementation . . . 104
6.1.3 Use Case Evaluation . . . 106
6.1.4 Threat-based Evaluation . . . 109
6.2 Helverify . . . 112
6.2.1 System Requirements . . . 113
6.2.2 Voting Protocol . . . 114

6.2.3 Architecture of Helverify . . . 119
6.2.4 Implementation . . . 121
6.2.5 Performance and Scalability Evaluation . . . 132
6.2.6 Trust Boundaries and Key Lengths . . . 136
6.2.7 Evaluation of Properties and Requirements . . . 139
6.2.8 Real-World Scenario . . . 140

7 Decentralized Remote Electronic Voting **144**
7.1 Provotum HE 3.0 . . . 145
7.1.1 System Design . . . 145
7.1.2 Voting Scheme . . . 147
7.1.3 Security Analysis . . . 153
7.1.4 Performance Analysis . . . 157
7.2 Verifiability Extension for Provotum HE 3.0 . . . 161
7.2.1 Verification and Accountability . . . 161
7.2.2 Comparison of Verification Mechanisms . . . 166
7.2.3 Design of Verifier Application . . . 166
7.2.4 Voter Front End . . . 171
7.2.5 Trust Boundaries of the Provotum System . . . 172
7.2.6 Implementation . . . 174
7.2.7 Analysis and Evaluation . . . 182
7.3 SSI-based IdM System for Provotum HE 3.0 . . . 190
7.3.1 Digital Identity Management . . . 191
7.3.2 Self Sovereign Identity (SSI) Primitives . . . 194
7.3.3 Provotum 3.0 Security Analysis . . . 203
7.3.4 Design of an SSI-based IdM System . . . 208
7.3.5 Implementation . . . 213
7.3.6 Evaluation . . . 221
7.4 ProvotuMn 3.0 . . . 228
7.4.1 Building Blocks . . . 228
7.4.2 System Design . . . 230
7.4.3 Voting Scheme . . . 231
7.4.4 Discussion of Properties . . . 236
7.4.5 Performance Evaluations . . . 240

7.5 Æternum . . . 246
7.5.1 System Design . . . 247
7.5.2 Implementation . . . 252
7.5.3 Performance Evaluation . . . 254
7.5.4 Security Analysis . . . 256

8 Summary, Conclusions, and Future Research **260**
8.1 Summary . . . 261
8.2 Review of Research Questions . . . 263
8.3 Future Research Directions . . . 266

References **292**

A Publications **293**
A.1 List of First Author Publications . . . 294
A.2 List of Co-Authorship Publications . . . 295

B Cryptographic Algorithms **298**

C Helverify Implementation Details **303**

D Provotum 3.0 HE **314**
D.1 Definitions . . . 314
D.2 Algorithms . . . 315

List of Figures **327**

List of Listings **331**

List of Tables **332**

Curriculum Vitae **334**

Computers are useless. They can only give you answers.

Pablo Picasso (1968)

1 Introduction

Historically, technology generally supported elections to ensure the correct organization and execution of electoral processes. The emergence of the internet and the omnipresent digitalization changed the fabric of societies and governments, naturally including voting systems. Thus, government services are becoming increasingly digitized [6], and a key effort is digitizing democratic processes [189], such as deploying Electronic Voting (EV) systems. The capability to execute fair and secure voting is a cornerstone of democracies and, at the same time, one of the most challenging processes in governments. Voting systems operate under stringent constraints and are designed and implemented with essential requirements, often contradicting one another. Trustworthy elections are essential to ensure long-term stability and allow people to participate in public affairs [305].

Advances in Information and Communication Technology (ICT) enable the application of New Voting Technologies (NVT), which can use ICT in any phase of the electoral process, not just voting itself [187]. NVT includes a range of technologies, from optical scanners and EV machines to simple spreadsheets for the tabulation of votes. From an election administration perspective, it is unthinkable to carry out voting and electoral processes without the use of NVT [187] because it is so widespread and omnipresent. However, these technologies must remain operationally efficient.

Still, security remains critical and trustworthy voting systems are crucial for public acceptance and, thus, the long-term stability of democracies.

Digitizing voting processes involves complex cryptographic mechanisms that require various domain experts to collaborate and contribute. Furthermore, REV research is highly interdisciplinary, involving applied cryptography, systems security, as well as legal and sociological aspects. Designing, implementing, and deploying voting protocols is challenging, especially since there are often conflicting system properties and practical complexities to overcome [52]. For that reason, security research on cryptography has a high impact on the real-world systems being developed and deployed. For instance, in Switzerland, a major REV system's deployment was halted because critical implementation errors were found after the source code had been published [181]. Also, expert discussions on cryptographic protocols directly impacted the Swiss regulations defining the minimum requirements for REV systems [81], thus aligning the legislation with the best practices from theoretical and applied research.

Voting systems present a unique dichotomy between privacy and verifiability, and the key challenge is achieving a private and verifiable electoral process [158]. The decisive aspect is whether a voting system can provide ballot secrecy to voters *without* leaking information while providing verifiable proof that convinces the public that the results are correct. In addition, the public's trust does not allow for any inexplicable (*i.e.*, unverifiable) results or errors, while the privacy of the ballot must be ensured indisputably. Also, it has been proven that achieving Unconditional Privacy (without any computational hardness assumptions) and Universal Verifiability (UV) at the same time is impossible [52]. Thus, achieving a favorable balance between privacy and verifiability is the fundamental goal of deploying electronic voting in the real world. The perfect balance of these properties depends heavily on the country's governmental structures (*e.g.*, electoral system, state of digitization) and, crucially, the public's acceptance and trust in the governmental entities.

Voting systems vary country by country, and so do the specific requirements for privacy and verifiability. The use of NVT in vote casting is globally heterogeneous (*cf.*, Figure 1.1). For instance, the United States, the Russian Federation, Belgium, and France use optical scanners and EV machines. However, the use of voter-facing technologies were stopped in Germany, the Netherlands, and Kazakhstan due to compromised systems and hacks, lack of public support, or court rulings [187]. Consequently, the adoption of NVT is also driven by social, economic, and political factors and not entirely by the technological (*e.g.*, security properties) arguments at hand.

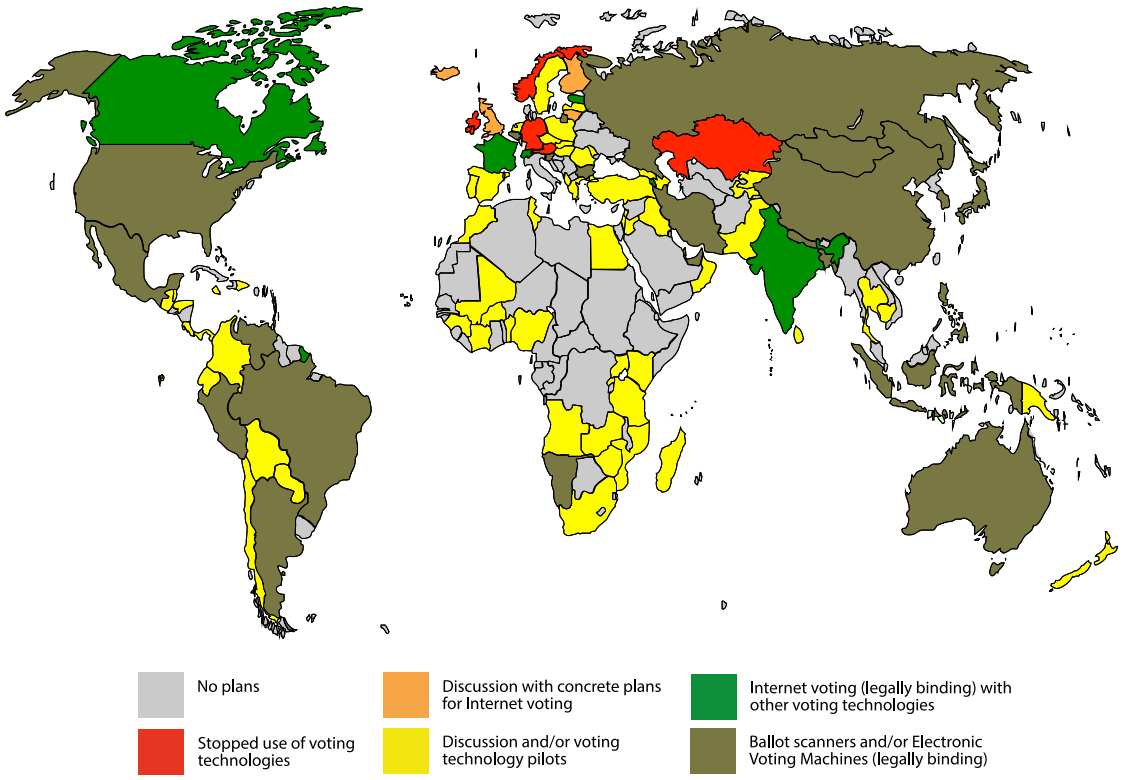


Figure 1.1: Global Map of REV Adoption per 2016, from [186]

1.1 Remote Voting Systems

A key difference and crucial aspect of voting processes is whether votes are cast in person (*e.g.*, in a booth) or remotely (*e.g.*, via the internet). Remote EV (REV) systems allow voters to cast their ballot remotely from an uncontrolled environment. Driven by the adoption of the internet and digitalization, several countries have started to deploy and test REV systems that allow voting in an uncontrolled environment [188]. Similar to Remote Postal Voting (RPV), an uncontrolled environment poses the potential threat of vote coercion [158], family voting, and simplified vote buying/selling [189]. For example, Estonia is a pioneer in NVT, offering an electronic identity and having introduced REV as an official legally-binding voting channel in 2005 [256]. Similarly, some cantons of the Swiss Federation offer REV to enable participation in the voting and electoral processes for citizens living abroad. Additionally, enabling REV is part of Switzerland's governmental strategies for pushing digitalization forward for public authorities and processes [83, 291].

From a practical perspective, remote voting systems inherently bear risks of voter coercion and vote-buying, and consequentially, these risks also apply to paper-based mail-in remote voting. Some authors argue that EV is by default worse than paper-based and in-person voting because EV comes with too many security threats [240]. However, mail-in remote voting is also prone to manipulation and tampering [180]. Therefore, using paper-based ballots by default does not render these voting systems more (or less) harmful than EV systems. For instance, the Swiss Remote Postal Voting (RPV) system bears risks and threats along the voting process [180] that fraudsters have exploited in recent years [34, 276]. However, in the Swiss case, the impacts of these threats are partially mitigated because the system is physically decentralized. Because of that, the threat detection and mitigation differs from REV. Regardless of which type of system (*e.g.*, paper-based, EV, or REV) is used, it is necessary to assess the security of voting systems explicitly with threat models, risk assessments, and penetration testing [96] to *(i)* identify threats and weaknesses and to *(ii)* mitigate them, and ultimately determine whether they are harmful or not for the election process. These assessments also highlight whether the privacy and verifiability properties align with the public's trust assumptions of the deployed voting system.

In the case of EV systems, a Public Bulletin Board ( ) serves as a tamper-proof audit trail. s store evidence of the correct execution of the voting protocol from the beginning to the end of the election. Thus, the immutability of s is a technical requirement to protect stored audit trails from tampering (*e.g.*, proofs, logs, and ballots). Additionally, the verifiability of the voting protocol is ascertained by ensuring that the verifiable proof remains authentic. The fundamental challenge of publicly releasing the data stored on s is ensuring that the published data (*e.g.*, ballots, proofs, and logs) does not leak any privacy-sensitive data and does not impose any risks on voters. Once the data is published, powerful adversaries can store information indefinitely and attempt to reveal the plain text of this data in the future. These long-term privacy concerns are crucial for the confidentiality and integrity of voting systems in general, but vital for EV systems, where the is publicly readable over the internet by anyone, which is a deliberate design choice to ensure transparency.

Only transparent, privacy-preserving, verifiable voting systems are accepted to conduct trustful elections [139]. Instead of relying on the honesty of government authorities and election officials, decentralized procedures can help mitigate the risks of centralized failures, sabotage risks, or the collusion of dishonest actors. Serving as a transparent, immutable, and distributed storage medium, Blockchains (BC) and Distributed Ledgers (DL) can offer new benefits for voting systems [67]. Instead of relying on a single centralized authority, a system of distributed and equivalent authorities can be deployed to emulate the physical decentralization of processes and authorities. In addition,

BCs and DLs can serve as a fully replicated and tamper-proof audit trail, enabling the verification of cryptographic proofs, which are crucial for verifiability properties [175].

This PhD thesis engages the tension between privacy and verifiability in decentralized voting systems, evaluating voting systems deployed in the real-world (*e.g.*, the Swiss RPV), while proposing novel approaches and protocols toward fully decentralized REV, with a focus on, but not limited to, the Swiss use-case. In essence, this PhD thesis addresses challenges in voting systems and presents solutions leveraging distributed systems (*e.g.*, BC and DL) in combination with applied cryptography to propose novel solutions towards private and verifiable voting systems.

1.2 Problem Statement

Recent studies [291] indicate that almost 70% of Swiss citizens welcome an EV system [222]. Despite the positive sentiment around EV, a political position proposed a moratorium on EV in Switzerland [89]. According to their initiative, a REV system has to be "*at least as secure as the current RPV system*". Thus, the fundamental question is: what exactly does such a minimal level of security involve? To the general public, EV often provokes a fear of change, partially based on presumptions that the current RPV system is mostly analog and tamper-proof. However, the current Swiss RPV system is already partially electronic, since many steps involve NVT (*e.g.*, electronic e-Identity management tools, e-Counting and internet-based result transmission tools). Thus, defining and comparing the security properties of a REV also requires an in-depth analysis of the current RPV system in Switzerland. Reducing costs and increasing voter turnout by providing a convenient way of voting are important considerations for and by Swiss authorities [209]. By 1994, all cantons accepted votes by postal mail. Today, RPV is the dominant voting channel, used by approximately 90% of the voters in Switzerland [121]. Only a remaining 10% choose to cast the ballot physically into the urn by visiting the Election Office. From a security perspective, casting the ballot into the urn physically is the maximum notion of Ballot Secrecy (BS) that can be achieved in the current Swiss RPV. Cantons and municipalities execute a degree of independence in decisions on how to handle certain parts of the voting process. Therefore, the current RPV system in Switzerland is neither universally documented or specified, nor homogeneous across entities.

The Swiss RPV is inherently built on ES and trusted relationships among the parties involved. For any eligible voter, the current process is hard to decipher and impossible to verify. In [180], (*i*) an overview of the Swiss RPV system, (*ii*) a detailed insight into the process flow, and (*iii*) a respective Risk Assessment (RA) was provided. The analysis in [180] provides critical Threat Events (TE) emerging during various stages of the voting process. Still, the current RPV system clearly presents

benefits too, such as its physical decentralization and distribution of trust. Due to the federal and decentralized structure of Switzerland, each canton and municipality manages their respective jurisdictional electoral procedures autonomously. Nevertheless, these benefits are undercut by using centralized information systems to administer or transfer crucial data (*e.g.*, Electoral Registers (ER) or web-based assistance tools to transmit intermediate results [285]). The deployment of a REV system potentially decreases the necessary amount of trust placed in governmental institutions and employees, shifting trust to transparent and verifiable processes instead [316]. For instance, various cryptographic mechanisms enable the electorate to verify different steps of these processes.

The RA in [180] identified critical TEs, which are based on real events. In 2011 [26], a re-count was impossible in Bern because some municipalities destroyed Voting Envelopes (VE) prematurely (*cf.*, Table 5.2, TE 14), instead of waiting for official validation by the Federal Chancellery (as required by the law). Additionally, physical storage of VEs often lacks secure access control and government employees can easily access cast VEs (*cf.*, Table 5.2, TE11). For instance, in 2015, an employee opened VEs, changed the ballots, and recast them [34]. In October 2019, in Glattfelden, a fire destroyed already cast VEs [320] and a small municipality in the canton of Vaude did not transfer their vote count until midnight, which delayed the final tally [105]. These incidents highlight the practical relevance of TEs identified in [180]. As these TEs illustrate, voting and electoral processes offer a large attack surface. Shifting focus to Swiss REV systems, the legal ordinance *i.e.*, VEleS [81] defined several new security and verifiability requirements. The core of every REV system is the underlying protocol; thus secure implementation is crucial. In order to establish public trust in a REV system, transparency is required [316]. In the example of the Swiss Post REV system, the code has shown critical vulnerabilities [133], insufficient transparency, and unclear documentation [184].

Based on these statements, four main issues of the Swiss RPV were identified: *i* heterogeneous and unverifiable processes [180], formed by *ii* centralized information system architectures in the Swiss RPV [180] and EV systems [133], which use *iii* insecure communication channels like telephone, fax, or non-secure email for transmission of preliminary results [285], and *iv* insufficient documentation and transparency in already-deployed REV systems [184]. With these key privacy and verifiability problems identified, this thesis proposes improvements to the existing Swiss RPV system, and addresses privacy and verifiability problems. Further, a set of sub-problems are tackled as well, such as *(i)* the lack of scalable, self-sovereign, and verifiable identity management for a decentralized system, *(ii)* long-term privacy concerns, as well as *(iii)* the decentralization of digital REV systems. With these problems identified, the Research Questions are posed and their respective research methodology is described in the next Section.

1.3 Research Questions

This PhD thesis engages the challenges outlined in the previous Sections, reinforcing the need for improvements to voting systems. The major focus of this PhD thesis lies within *(i)* the dichotomy of privacy and verifiability of voting systems, *(ii)* the centralization of voting systems, as well as *(iii)* the broader privacy and verifiability trade-offs and challenges, and how they can be addressed in practice. Within that context, four key Research Questions (RQ) were posed, guiding the structure and content of this thesis. Therefore, this PhD thesis is guided be the following RQ.

RQ1 How can existing electoral processes (*e.g.*, the Swiss RPV) be quantified and compared to novel voting channels (*e.g.*, a REV system), considering trust assumptions, privacy, and verifiability properties, and the risks involved in both systems?

Methodology RQ1 Process Analysis, Risk Assessment

Work on RQ1 assesses the current electoral processes deployed in Switzerland, carefully considering the procedural heterogeneity due to the Swiss federal structure. RQ1 will use a qualitative method to compare electoral processes. A risk assessment provides an overview of the relevant TEs, with the focus on confidentiality (C), integrity (I) and availability (A), as defined in [299]. Thus, RQ1 analyzes the current Swiss electoral processes, enabling a comparison between alternative voting channels (*e.g.*, comparing RPV with REV). Investigations of RQ1 build the foundation for all subsequent RQs.

RQ2 How can electoral processes (especially the Swiss) be publicly verifiable, while retaining the privacy of the ballot?

Methodology RQ1 Prototype Design, Implementation, Evaluation of Secret Ballot Voting Protocols

RQ2 is at the core of fair and secure secret ballot voting schemes and REV research. Therefore, RQ2 investigates applied cryptography and secret ballot voting schemes to provide privacy-preserving and verifiable electoral processes, considering trust assumptions, security risks, and threats identified in RQ1.

RQ3 How can trust be decentralized in systems used for voting and electoral processes while remaining legally compliant and economically viable?

Methodology RQ3: Qualitative Comparison and Evaluation of different BC and DL platforms for electoral processes to distribute trust

Once current systems are centralized to a certain degree, the verifiability of votes can be compromised, forcing the electorate to trust a centralized process or system operated by Voting Authorities (VA) or ES. Thus, work on RQ3 investigates methods to decentralize trust in centralized voting and electoral processes by leveraging advances in distributed systems (*e.g.*, BCs and DLs). RQ3 relates directly to RQ1 and RQ2 because distributing trust influences privacy and verifiability properties. Additionally, legal requirements in each canton, as well as the federal legislation, are considered. Furthermore, the economic viability is investigated, since cost also determines an important factor in public services.

RQ4 Which architectural components are required to provide privacy and verifiability for voting and electoral processes?

Methodology RQ4: Prototype Design, Implementation and Evaluation of a BC-based REV system, combining previous RQs

RQ4 relates directly to methods of guaranteeing a decentralized process, able to satisfy trade-offs defined in all previous RQs (1-3). The major characteristic of a BC-based system is the transparency, recording an immutable and verifiable audit trail. However, this transparency conflicts with privacy properties. Therefore, by applying cryptographic building blocks, acceptable and legal Ballot Privacy will be reached while simultaneously ensuring individual and universal verifiability properties.

1.4 Thesis Outline

Although this thesis frames research questions with the Swiss legislation and voting systems in mind, the research contributions are generalizable. Overall, this thesis drives forward decentralized voting systems' state of the art. This PhD thesis is organized in seven chapters as indicated in Figure 1.2. **Chapter 2** forms the theoretical foundations, including the cryptographic fundamentals necessary for the this thesis. Voting schemes require sophisticated cryptographic protocols and algorithms to achieve privacy and verifiability properties. **Chapter 3** outlines the essential requirements for REV systems. These requirements are based on the protocol (*e.g.*, privacy and verifiability), or general system requirements. Also, this chapter details the evolution from centralized to decentralized REV systems, outlining essential building blocks for voting systems, including architectural models for Public Bulletin Boards (PBB). Further, REV approaches are overviewed and described, so the foundational

understanding for the different schemes is laid out. Then, **Chapter 4** contains related work on *(i)* cryptographic voting schemes and *(ii)* BC- and DL-based voting systems. A literature review is performed to map systems and schemes and their respective properties to achieve a complete overview.

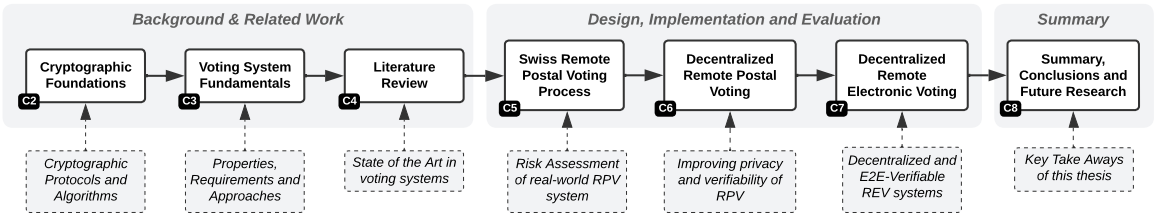


Figure 1.2: Structure of PhD Thesis

Next, **Chapter 5** contains detailed information on the current Swiss Remote Postal Voting (RPV) procedure, presenting (*i*) an overview of the Swiss RPV system, (*ii*) detailed insight into the process flow, and (*iii*) a respective risk assessment with threat events identified for each stage of the process. **Chapter 6** introduces Proverum, an approach for decentralized identity management in the Swiss RPV, where a dedicated DL is used to achieve Self-Sovereign Identity (SSI) management on a federal, cantonal and municipal level. Further, this chapter includes Helverify, a novel RPV approach tailored to the Swiss RPV, introducing End-to-End Verifiability (E2E-V) and Ballot Secrecy that does not rely on administrative processes. After that, **Chapter 7** introduces multiple decentralized REV systems (Provotum 3.0 HE, ProvotuMn, and Æternum). All of these systems provide solutions towards private and verifiable REV, including design decisions and architectures of each system. Further, implementation and technical details are outlined, followed by quantitative and qualitative evaluations. Finally, **Chapter 8** concludes this PhD thesis with a summary, followed by a detailed review of each research questions, outlining the contributions and conclusions for each of them. Finally, this thesis concludes with future research directions in REV, RPV and voting systems in general.

Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful.

—George Box and Norman Draper in "Empirical Model-Building and Response Surfaces" (1987)

2 Cryptographic Foundations

Understanding modern cryptography requires knowledge of its essential building blocks, principles, and assumptions. Therefore, this chapter focuses on the primitives necessary to understand advanced cryptographic voting protocols and their inner workings, *i.e.*, how and under which assumptions these protocols achieve their properties. This chapter uses color coding for cryptographic algorithms where elements in red are secret and private, while items in blue are public information and are known to everyone [278]. For a complete and in-depth overview of cryptographic primitives, please refer to [31, 278]. For application-centric descriptions of cryptographic protocols, please refer to [10, 164, 326].

Figure 2.1 depicts an overview of the most crucial cryptographic primitives associated with their respective REV approaches (which will be covered in Chapter 3). Along Chapter 2, the cryptographic primitives are introduced with increasing levels of complexity. It should be noted that the associations do not define an *exclusive* usage of the respective primitive in the associated REV approach. For instance, Public Key Cryptography is not used exclusively in the Encrypt & Sign approach but used in all other techniques as well.

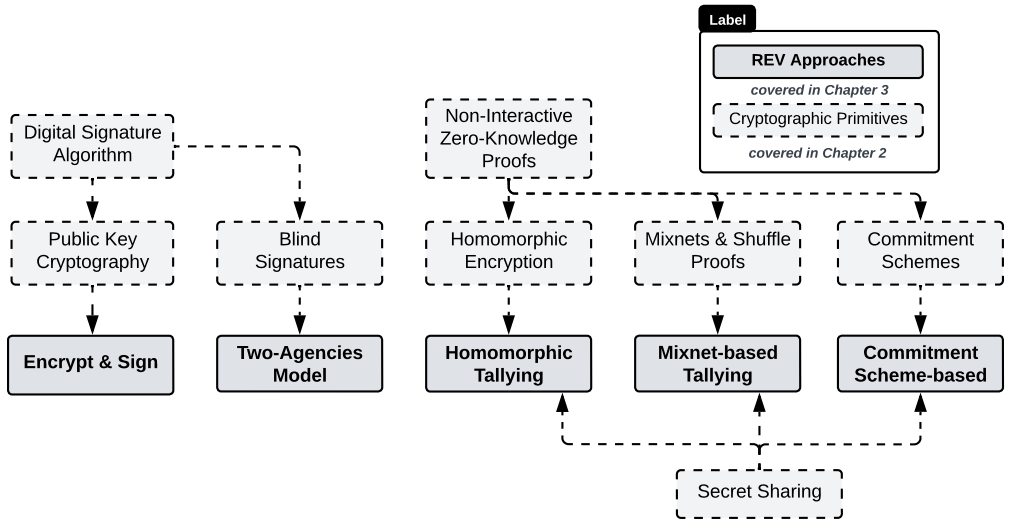


Figure 2.1: Overview of Major Cryptographic Primitives (introduced in Chapter 2), in Their Relation to REV Approaches (introduced in Chapter 3)

2.1 Cryptographic Fundamentals

Encryption is the essential application of cryptography because it enables keeping data and messages confidential. Encryption algorithms, so-called ciphers, use a secret value, called a secret key ( ) to encrypt plaintext to ciphertext [10]. Plaintext refers to the unencrypted message, whereas ciphertext refers to the encrypted message. Therefore, a cipher consists of an encryption and a decryption algorithm. There are two major types of encryption: symmetric and asymmetric. Symmetric encryption is the simplest form of encryption, where the same secret key ( ) is used to encrypt and decrypt (*cf.*, Fig. 2.3). The resulting ciphertext makes it impossible to learn any bit of information from the encrypted message, especially for attackers.

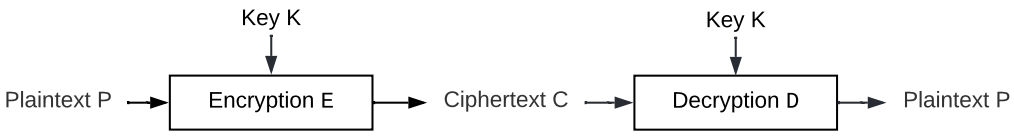


Figure 2.2: Basic Symmetric Encryption and Decryption Algorithms

Asymmetric encryption (also referred to as public-key encryption) uses two different keys, a public key ( ) and a secret key ( ), which is often referred to as the *private key*.Suppose Alice wants to send a message to Bob. In order to do that, Alice will encrypt the plaintext with Bob's public key ($_{Bob}$), which Bob decrypts by using his secret key ($_{Bob}$).

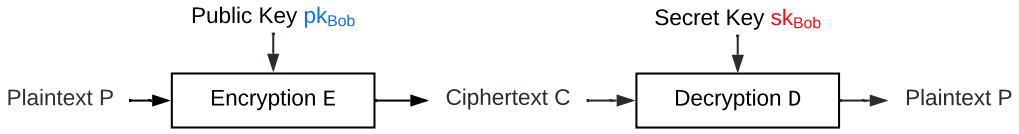


Figure 2.3: Basic Asymmetric Encryption and Decryption Algorithms

The secret key's confidentiality and security are crucial in symmetric and asymmetric encryption. Publicizing algorithm specifications is considered standard procedure nowadays, while in the past, both the secret keys and cryptographic algorithms were kept secret [10]. Back in 1883, Auguste Kerckhoff had already formulated requirements necessary for a military-grade encryption system [165, 166]. One requirement states that the system must *not* require secrecy and could be stolen by the enemy without resulting in trouble. Hence, the *security through obscurity* approach is generally insufficient and not advisable [164]. Past and present examples show that cryptanalysis highlights weaknesses of cryptographic algorithms. Therefore, the next section covers different security goals and attack and threat models that influence secure and modern cryptography.

2.2 Cryptographic Security

Although ciphers have been broken and proven to be insecure in the past, there was no agreed-upon framework of what a *secure* cipher should entail. Cryptographic security is quantifiable, meaning that it is possible to calculate the effort needed to break a particular cryptographic algorithm, assuming a secure technical implementation of the algorithm is provided. Thus, the quantification is an exact measurement of the computational effort needed to break the algorithm [10].

2.2.1 Security Goals and Attack Models

In cryptography, there are two major *security goals*: *(i)* Indistinguisability (IND), which means that ciphertexts should be indistinguishable from random strings, and *(ii)* Non-Malleability (NM), which

means that given ciphertext c_1, it should be impossible to create another ciphertext c_2, whose corresponding plaintext p_2 is related to p_1 in a meaningful way. These *security goals* are then combined with *attack models* (*cf.*, Fig. 2.4). Hence, security notions are described as combinations, *e.g.*, IND CPA, IND CCA.

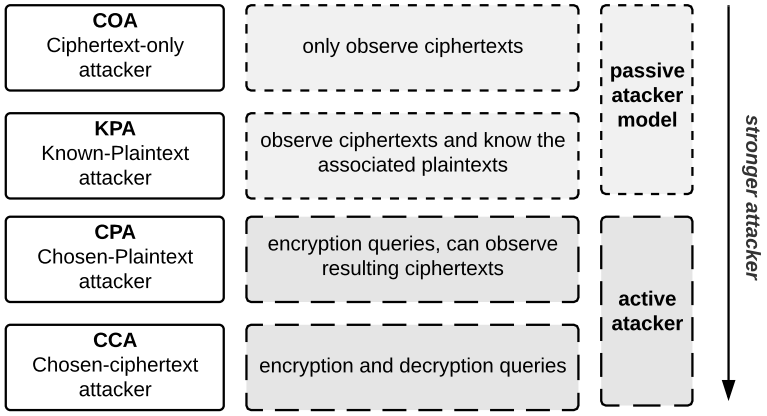


Figure 2.4: Attack Models and Attacker's Capabilities

The most fundamental security notion is IND CPA, also referred to as *semantic security*, which means that ciphertexts should not leak any information about plaintext as long as the secret key is secret [10]. [116] proved the equality of polynomial and semantic security, whereas [114] extended these notions, proving that deterministic asymmetric encryption can not be semantically secure, since it can not be indistinguishable [103] *i.e.*, the adversary knows the encryption function, and can compute the ciphertext corresponding to each plaintext. One way to achieve IND-CPA is by using randomized encryption functions. Encryption is then expressed as c r m where r is fresh random bits. However, it is crucial that the decryption remains deterministic.

Randomness is an essential element in cryptography because without it, cryptographic operations would become predictable and thus insecure [10]. Without the ability to generate random secret values, secret key generation would not be possible, and even further, semantic security is not possible without a source of randomness [11].

2.2.2 Hard Computational Problems

Computationally hard problems (*e.g.*, factorization of prime numbers as defined in RSA) are the backbone of modern cryptography. In essence, these problems are easy to formulate but extremely hard to solve. For computationally hard problems, no algorithm will run in a reasonable amount of time,

and they are also often referred to as *intractable* problems [10]. Most REV systems use cryptographic primitives, which are in turn based on hard problems. BCs and DLs are also based on modern cryptography and, thus, are also based on such intractability assumptions. It is crucial to assess these problems against current and future adversaries, especially with the advent of quantum computing, which may break certain hard problems. Considering BCs and DLs, data persists on publicly readable storage, which means that future adversaries could store that data now and break the underlying schemes in the future (*e.g.*, store election data now and break ballot secrecy in the future). Therefore, these assumptions impact the long-term privacy and verifiability properties directly because they often assume hard problems. Hence, it is important to define the assumptions and hard problems that are considered by these primitives and the direct implications on security notions.

Most voting schemes are based on public-key encryption methods and are thus based on assumptions of computational complexity [158]. Since every major public-key cryptosystem deployed today relies on either factoring (using the RSA algorithm) or the Discrete Logarithm Problem (DLP), systems using these schemes inherit the underlying assumptions. One of the key hard problems that modern cryptography is built on is the *Factoring Problem*, which is defined as finding the prime numbers p and q, given a large number $N \quad p \quad q$. The widely known and used RSA algorithms are based on the factoring problem. The security of the factoring problem is based on the fact that factorization of prime numbers is hard, *i.e.*, it is hard to decompose N as a product of prime numbers.

The *Discrete Logarithm Problem (DLP)* is similar to the factoring problem. DLP describes a *discrete* problem, which means it deals with integers, as opposed to real numbers (which would be called continuous). Further, it deals with a *logarithm*, *i.e.*, the inverse function of exponentiation. Also, it deals with mathematical groups of prime order modulo p, which are a set of elements (typically numbers) that are related to each other according to well-defined rules written as Z_p. The DLP is defined as finding the y for which $g^y \quad x$, given a base number g within some group Z_p, where p is a prime number, and given a group element x.

The last group of hard problems is the Diffie-Hellman problems, which are a special variation of the DLP. The *Computational Diffie-Hellman (CDH)* problem is also a variation of the DLP. The CDH describes computing the shared secret g^{ab} given only the public values of g^a and g^b, and not any of the secret values a or b. If an eavesdropper was to obtain g^a and g^b, it should not be possible for her to determine the shared secret g^{ab}. Lastly, the *Decisional Diffie-Hellman (DDH)* problem is stronger than the CDH. The DDH assumes that the attacker is not able to compute g^{ab} partially, given the values of g^a and g^b, *i.e.*, the attacker can not learn *anything* about the shared secret. To ensure that, the value of g^{ab} needs to be indistinguishable from a random group element.

The importance of these problems may not be obvious at first glance, but most of the cryptographic methods used in everyday communications and the internet are based on using these hard problems. Therefore, the problems are *assumed* to be hard to solve. These assumptions build the foundation of all major infrastructure used in modern computer networks and in many voting protocols and REV systems, as well as BC and DL networks. The implications of this are clear: REV systems, BCs, and DLs alike base their security on public-key cryptography and hash functions. These primitives are used to authenticate transactions (or ballots) with signatures and are also used in communication channels and consensus algorithms to identify authorities (*e.g.*, PoA). Collision-resistant hash functions are used to link together blocks of transactions, effectively forming the chain or ledger.

Advances in quantum computing threaten these building blocks, since it is estimated that in the next 20 years, a powerful Quantum Computer (QC) will be powerful enough to break current strong public-key cryptosystems by breaking the underlying hardness assumptions [99]. The research community is tackling these issues with many post-quantum cryptography efforts and various novel approaches for post-quantum encryption and signatures. For example, the NIST is conducting standardization processes to establish solid post-quantum public-key cryptography mechanisms [229].

2.3 Public-Key Cryptography

Public-key cryptosystems (*i.e.*, asymmetric encryption) require a key pair consisting of two distinct keys, a public ( ) and a private key *i.e.*, secret key ( ). In Public-Key Cryptography (PKC), the public key is used for encryption and digital signature verification, while the private key is used for decryption and creating digital signatures [293].

A public key cryptosystem consists of the following components: a plaintext message m, an encryption algorithm ( ), a public/private key pair generated by a key generation algorithm ( ), an encrypted representation *i.e.*, ciphertext c and a decryption algorithm ( ) [293] (*cf.*, Equation 2.1).

$$m \qquad c \qquad m \tag{2.1}$$

In 1977, the Rivest-Shamir-Adleman (RSA) cryptosystem became the first public-key encryption scheme. It took many decades to standardize and safely develop secure RSA implementations [10].

Another prominent public-key cryptosystem is ElGamal [95], introduced in 1985. In the context of electronic voting, ElGamal has crucial advantages compared to RSA, e.g, ElGamal is based on the Decisional Diffie-Hellman (DDH) assumption and not on the RSA problem. Further, ElGamal is a homomorphic scheme, which only holds for RSA if it is used without padding [203], *i.e.*, adding

good padding (*e.g.*, OAEP) to RSA will render it semantically secure, but the scheme loses its homomorphic property [10].

2.3.1 ElGamal Public-Key Cryptosystem

The ElGamal public-key cryptosystem was formulated in 1985 [95]. ElGamal is often used in cryptographic voting schemes because of the multiplicative homomorphism, which enables the encoding of votes in ciphertexts (*cf.*, Section 2.3.1). Further, ElGamal offers semantic security (*cf.*, Section 2.3.1) because ciphertexts are not deterministic, but probabilistic. The computational hardness assumptions for ElGamal are based on the Decisional Diffie-Hellman (DDH) problem (*cf.*, Section 2.3.1). The DDH assumption is commonly considered to hold a safe prime in the subgroup of quadratic residue modulo: $p \quad 2q \quad 1$ [30].

ElGamal is defined within the cyclic group . The cyclic group, $_q$ with prime order q, is defined over a finite field $_p$, defined over positive integers modulo p, where p is an odd prime. The cryptographic operations are performed in a subgroup $_q \quad _p$, a multiplicative cyclic group $_q$ defined over the finite field $_p$. Alternatively, the cyclic group can also be defined over an elliptic curve that is itself defined over a finite field [95].

In ElGamal, the public elements are $p \ q$ and two independent generators g and h, where $g \ h$ $_q$ 1. The private key is a random value: $_r \ _q$, where $q \quad \frac{p \ 1}{2}$. The public key is defined as follows:

$$g^{sk} \mod p$$

The encryption of a plaintext message m is performed by using a random value $r \ _r \ _q$. Thus, each message has a different ephemeral key r, which means that encrypting the same message m twice will produce different ciphertexts, ensuring semantic security.

$$E \ m \quad c \quad a \ b$$
$$g^r \mod p \quad ^r \ m \mod p$$

The decryption of a ciphertext $c \quad a \ b$ is performed by computing the multiplicative inverse of a^{sk} multiplied with b.

$$
\begin{aligned}
D\ c \qquad m \quad & a^{\mathsf{sk}} \ ^{1}\ b \mod p \\
& g^{r^{\mathsf{sk}}} \ ^{1} \quad ^{r}\ m \mod p \\
& g^{r^{\mathsf{sk}}} \ ^{1}\ g^{\mathsf{sk}^{r}}\ m \mod p \\
& g^{r\ \mathsf{sk}} \ ^{1}\ g^{r\ \mathsf{sk}}\ m \mod p
\end{aligned}
$$

Message Encoding in ElGamal

Using the correct message encoding is crucial for voting systems because votes (*i.e.*, messages) are first encoded and then encrypted. In detail, messages in the ElGamal cryptosystem must be encoded, *i.e.*, mapped with a reversible function to a group element, such that the DDH assumption holds [65, 94]. Otherwise, the ElGamal cryptosystem is not secure against plaintext attacks (*i.e.*, does not achieve `IND-CPA`). Hence, two major message encoding variants are possible [91].

The first variant to perform a message encoding was introduced in [65]. The core premise is to encode any message m $_{q}$ as g^m. The corresponding ciphertext is given by $E\ m$ c $a\ b$ g^r ^{r} g^m . The encoding enables additive homomorphic operations, making the ElGamal cryptosystem *additive* instead of *multiplicative* homomorphic. A downside of this encoding is that the plaintext, resulting from the decryption, must be decoded by computing the discrete logarithm of g^m (*i.e.*, g g^m), and computing discrete logarithms is hard. It is only possible for a restricted message space m o 1 where brute-forcing or using the `baby-step-giant-step` algorithm is possible [94]. Below, two ElGamal ciphertexts $E\ m_1$ and $E\ m_2$ are summed homomorphically. To achieve that, the components are multiplied, *i.e.*, its exponents are added up, resulting in the equivalent sum as though the plaintexts m_1 and m_2 had been added together.

$$
\begin{aligned}
& m_1 \quad m_2 \qquad\qquad m_1 \qquad\qquad m_2 \qquad\qquad g^{r_1\ r_2} \quad ^{r_1\ r_2} \quad g^{m_1\ m_2} \\
& \qquad m_1 \quad m_2 \qquad\qquad\qquad\qquad m_1 \quad m_2
\end{aligned}
$$

The second encoding variation ensures that all messages are restricted to quadratic residues in a safe prime group of order q. Using this approach, it is necessary to check that the equality: m^q mod p 1 holds *before* a message is encrypted. If the equality does not hold, the message must not be encrypted [94]. A downside of the second variant is that the equality check needs to be performed *before* encrypting, thus increasing the computational overhead due to the modulo exponentiation. However, in real-world practical systems, the increase is negligible for a single operation.

2.3.2 Digital Signatures

Digital signatures are the result of the cryptographic transformation that allows for verification of origin authentication proof, data integrity, and non-repudiation [167] *i.e.*, that a message really originates from a known sender and that the message was not manipulated along the way. While encryption schemes provide protection against eavesdropping and ensure confidentiality, digital signatures allow message authentication and integrity. In other words, a signature scheme allows somebody (*e.g.*, Bob) to sign a message m, so that a signature can be verified by anyone. This verification step proves that the message could only have been signed by Bob himself, or by someone who was in possession of Bob's private key $_{Bob}$.

A digital signature scheme consists of a key generation ( ), a signature ( ), and a verification ( ) algorithm (*cf.*, Equation 2.2). A signature is valid if returns b 1.

$$\sigma \qquad m \qquad (2.2)$$
$$b \qquad m \;\; \sigma$$

2.3.3 Blind Signatures

A Blind Signature (BS) is a cryptographic scheme that enables a user to request a digital signature of a message while keeping its content secret to the signer. In practice, this can be employed to ensure that a signer cannot trace the usage of its signature. [49] first proposed BS to enable untraceable electronic mail and payments. BS find application in early REV schemes as well [104], which would work as follows: Suppose a voter v encloses her $ballot_v$, which is written on carbon paper, in an envelope and seals it. The sealed envelope is given to a trusted registrar r. The registrar verifies *(i)* the identity and eligibility of v and *(ii)* verifies that v has not cast a ballot yet. Then, r signs the envelope, which includes the ballot on the carbon paper, and returns it to v. Now, the voter v can *unblind* the ballot and cast the signed carbon paper $ballot_v$ separately. In short, the use of BS [49] allows a registrar r to sign a message m without knowing its contents [158].

BS schemes exist for many public key cryptosystems (*e.g.*, for RSA). We will define a BS scheme as per [28] and [104], defined by the following tuple of algorithms: key generation ( ), Blinding ( ), Signing ( ), Unblinding ( ), and verifying ( ) [49]. shall be deterministic, while the rest may be randomized.

outputs a keypair vk . The blinding algorithm takes a message m and a verification key vk and outputs a blinded message b, as well as an unblinding factor u: $b \;\; u$

m vk The signing algorithm takes a signing key and a blinded message b and outputs a blinded signature s b sk . The unblinding algorithm takes a verification key , a blinded signature s and an unblinding factor u and outputs a signature σ: σ vk s u . Finally, the algorithm takes a verification key , a message m and a signature σ and returns a bit v m σ . The bit is 1 if the signature verification succeeds, else v equals 0.

2.4 Commitment Schemes

Commitment Schemes (CS) are another essential building block for cryptographic protocols. A CS is used to commit to a value v without revealing it at the time of commitment, but possibly revealing it later *i.e.*, a CS is useful when a prover ( ) must be prohibited from changing a value v they have committed to, but v should also stay private at the time of commitment [278].

Given a commitment key c, which is usually a random string chosen by the prover, the main property of a CS is [158]:

$$_c \quad _c\, v \quad v$$

Further, a CS follows the following properties [164]:

- *Hiding:* The commitment reveals nothing about v, *i.e.*, it should completely hide the committed value to any party involved.
- *Binding:* It is computationally infeasible to commit to a value v, but open the commitment to another value v , *i.e.*, the commitment truly commits the to the v.

The binding property can also be achieved by applying regular encryption to commit, and then later revealing the encryption key [158]. However, by nature, encryption schemes based on computational hardness assumptions are only computationally hiding. Thus, given enough resources, the decryption key can be revealed. However, the Pedersen CS [244] can be perfectly hiding, *i.e.*, no amount of computation can evaluate which value v was committed to.

Pedersen Commitment Scheme

The Pedersen CS [244] requires a prime-order subgroup $_q$ of $_p$ that fulfills the Discrete Logarithm (DL) assumption (*e.g.*, a Schnorr group). The commitment to a value v $_q$ is defined as

$$_p \quad r \quad v \qquad g^r h^v$$

where $r \in_R \mathbb{Z}_q$, *i.e.*, it is randomly chosen from $\mathbb{Z}_q$, and g and h are generators of $\mathbb{G}_q$. The value $c = \mathrm{com}_p(r, v)$ is published while r and v are kept private. The commitment can be opened by simply publishing the two values v and r. The receiver of the values then calculates $c' = \mathrm{com}_p(r, v)$ and accepts if $c = c'$ [244].

The scheme can be extended and used to commit to multiple values at the same time. A commitment to values $u_1, \ldots, u_n \in \mathbb{Z}_q$ is defined as

$$\mathrm{com}_p(r, u_1, \ldots, u_n) = g^r h_1^{u_1} \cdots h_n^{u_n}$$

with $r \in_R \mathbb{Z}_q$ and $g, h_1, \ldots, h_n$ being generators of $\mathbb{G}_q$ [203].

The Pedersen CS is binding and perfectly hiding. The binding property is fulfilled because of the DL assumption, *i.e.*, it is hard to find values r and v such that $\mathrm{com}_p(r, v) = \mathrm{com}_p(r', v')$. It is perfectly hiding because $c = \mathrm{com}_p(r, v)$ does not reveal any information about the committed value v. Although the hiding property does not depend on any computational hardness assumption, it holds because many values v' and r' can open the commitment c. An attacker can, therefore, not determine which v' and r' are the right ones [203].

2.5 Secret Sharing

The assumption of trusting a single party holding the private key material can not be considered practical and robust [203]. For instance, in a voting system, if a single party held the private key material for the election key, the holder is fully trusted to behave honestly, but could decide to decrypt any ballot at any time. Further, the trusted keyholder could decide to just destroy the private key and not decrypt votes at all, maybe after learning that the final tally after decryption was not as they wished. To mitigate these issues, cryptographic protocols for Secret Sharing were established.

Shamir's Secret Sharing (SSS) is one of the first methods used to share secrets, *i.e.*, private keys [274]. Unfortunately, with SSS, one single key holder, the so-called dealer, will know the full secret key ( ). In Shamir's k n threshold scheme, the dealer hands out keyshares to n trustees, of which a minimum threshold of k trustees need to collude to recreate the . Thus, SSS does not solve the problem of a single party holding all the information needed to break privacy.

In detail, SSS allows for dividing a secret key across a set of participants n. Thus, is split into multiple shares $_i$, and is encoded as the first coefficient in the k 1 -degree polynomial, which is defined over a finite field , such that P 0 . Since any polynomial of degree k 1 is uniquely

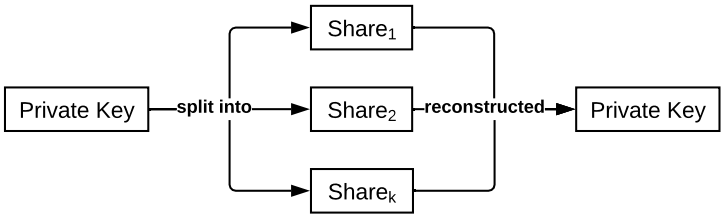


Figure 2.5: Secret Sharing

defined by k points, any arbitrary coalition of k trustees are able to reconstruct the polynomial P through Lagrange interpolation, and thus, recover the secret key P o [274].

A widely known extension of SSS is Feldman's Verifiable Secret Sharing (FVSS). The scheme is based a polynomial P that is computed over $_q$. Again, a dealer, which knows the secret key , distributes the points on the polynomial (*i.e.*, P 1 P 2 P N mod q) privately to the trustees [98]. To make the points verifiable by the trustees, the dealer broadcasts commitments with the coefficients publicly, allowing each trustee to verify the validity of its share.

2.5.1 Distributed Key Generation

In Secret Sharing schemes [29, 274], a dealer generates a secret s and distributes its shares s_i among the trustees. Improving on that, Distributed Key Generation (DKG) schemes do not require a trusted party [163]. Pedersen proposed an extension of FVSS in [243], where the secret value s remains completely unknown to any party, and the extension can be used by k authorities to select a public key , such that the corresponding secret key remains unknown, but can be constructed by a coalition of t authorities.

Unlike the dealer in FVSS, Pedersen's DKG [243] does not have a Single Point of Failure (SpoF). In this protocol, every participant runs a FVSS scheme simultaneously as a dealer. Each participant i generates random secret s_i and constructs a separate random polynomial P_i x , such that P o s_i. The shares are distributed and the commitments broadcasted by each participant. The secret value s is the sum of the properly shared s_i's:

$$s \qquad P_i \text{ o}$$
$$i \quad 1$$

Since FVSS has the additional property of revealing $y_i = g^{s_i}$, the public value y is the product of the y_i's that correspond to the properly shared s_i's [243].

In PKC, DKG presents an excellent way to achieve a dealerless threshold scheme, allowing decryption and signatures to be handled in a decentralized fashion. Unfortunately, a key problem in

DKG protocols is the assumption of a synchronous network, which is unrealistic given the internet's asynchronous nature [163].

2.5.2 Cooperative Decryption

Cooperative decryption depends on the secret sharing that was used initially to set up the cryptosystem. For instance, if there is a $(k \;\; n)$ threshold scheme, or $(n \;\; n)$ scheme, the algorithm differs. This thesis focused on an n out of n approach. The algorithm is outlined below.

The logical counterpart to the DKG protocol is the Cooperative Decryption of ciphertexts encrypted with the joint public key. Similar to the DKG protocol itself, each participant i must generate a set of decryption shares d_{ij} which will jointly result in successful decryption.

Step 1 Each participant i (*e.g.*, voting authority) partially decrypts the ballots c using its private key sk_i, producing a set of decryption shares d_i. These shares do not reveal anything about the plaintexts (*cf.*, Eq. 2.3).

$$\begin{aligned} &\text{input} && \text{ciphertexts: } \mathbf{c} && c_1 \;\; c_2 && c_j && c_m \;\; c_j && a \;\; b \\ &\text{output} && \text{partial decryptions: } d_i && d_1 \;\; d_2 && d_j && d_m \\ & && d_j && a_j^{sk_i} \mod p && && && (2.3) \\ & && && g^{r_j \; sk_i} \mod p \end{aligned}$$

Step 2 The decryption shares of all participants are combined to retrieve the combined decryption shares (*cf.*, Eq. 2.4) [126]. This step can be performed online and, therefore, can be verified by anyone, thus not requiring an proof.

$$\begin{aligned} &\text{input} && \text{partial decryptions: } \mathbf{D} && d_{ij} && {}^{N \;\; M}_{q} \\ &\text{output} && \text{combined partial decryptions: } d && d_1 \;\; d_2 && d_j && d_m \\ & && d_j && \prod_{i \;\; 1}^{N} d_{ij} \mod p && && (2.4) \end{aligned}$$

Step 3 The combined decryption shares are used to retrieve the decrypted ballots (*cf.*, Eq. 2.5) [126]. Depending on the message encoding (2.3.1), the decrypted ballots might still have to be decoded to

reveal the plaintexts [126]. Again, this step can be performed online and is therefore publicly verifiable by anyone.

$$
\begin{array}{lll}
\text{inputs} & \text{ciphertexts: } c & c_1\ c_2 \quad c_j \quad c_m\ c_j \quad a\ b \\
& \text{combined partial decryptions: } d & d_1\ d_2 \quad d_j \quad d_m \\
\text{output} & \text{plaintexts: } m & m_1\ m_2 \quad m_j \quad m_m \\
& & m_j \quad \frac{b_j}{d_j} \mod p \\
& & {}^{r_j}\ m_j \quad g^{r_j \mathsf{sk}}\ {}^{1} \mod p
\end{array}
\tag{2.5}
$$

2.5.3 Operational Concerns

Maintaining essential management procedures and solutions is crucial to assure Operational Security (OpSec). In [11], key management is described as: "The single hardest problem in cryptography. Key management won't be solved by quantum computers or with an NP oracle." Unsurprisingly, in practice, the primary concern in cryptographic protocols is to keep the safe. Voting authorities must take organizational security precautions for cryptographic key storage. For example, these countermeasures may require physically air-gapping the storage of the to a device that never connects to any network (*e.g.*, for high-stakes elections).

2.6 Homomorphic Encryption

Homomorphic Encryption (HE) allows operations on encrypted data. Therefore, confidential data remains private, even when a computation is performed in an untrusted environment, *e.g.*, a public cloud provider [8]. This property is used to tally votes in cryptographic voting protocols. Thus, every eligible voter can encrypt his or her vote using the public key ( ) for the election, resulting in an encrypted vote. No individual votes need to be decrypted because these votes can be added together homomorphically [158]. In detail, HE is based on a homomorphism (*i.e.*, a structure-preserving mapping) existing between plaintext and a ciphertext computation. The mapping can be defined as follows:

$$
E\ m_1\ pk \qquad E\ m_2\ pk \qquad E\ m_1 \quad m_2\ pk \tag{2.6}
$$

As indicated in Equation 2.6, the equality means that the ciphertexts (m_1 m_2) can be combined without using any secret key material. The specific operators and are arbitrary and depend on the underlying cryptosystem.

Suitable cryptosystems that provide a homomorphism partially under a single operation are ElGamal [95] (*cf.*, Subsection 2.3.1), the Paillier cryptosystem [235], and RSA [254]. However, ElGamal and Paillier are the most popular choice for REV schemes because both are non-deterministic, which is one reason why RSA, although offering homomorphic properties, can not be used for REV systems. A non-deterministic (*i.e.*, probabilistic) cryptosystem provides the security notion of IND-CPA, *i.e.*, IND-CPA combines the security goal of indistinguishable ciphertexts in combination with the chosen plaintext attacker-model, as further explained in Section 2.2.

2.7 Re-Encryption

Re-encryption is a technique that can be used in probabilistic encryption systems to change the ciphertext without changing the respective plaintext. Since the ciphertext depends on an independent random value r_1, its appearance can be altered by encrypting it again using a different independent random value r_2. The resulting ciphertext is the same as if it only depended on a single independent random value r_3. It is important to note that both and g need to remain the same throughout all encryptions [158].

GENERIC RE-ENCRYPTION

$$r_2 \quad r_1 \; m \qquad\qquad r_3 \; m \tag{2.7}$$

Re-encryption is used in REV systems to break the link between the voter and her ballot since re-encryption only changes the appearance of the ballot but not its content. The re-encrypted ballot can no longer be associated with the voter while the decrypted content (*i.e.*, the vote) remains unchanged. In all probabilistic homomorphic encryption system, re-encryption is supported by homomorphically adding an encryption of 0 (*cf.*, 2.8) or by multiplying with an encryption of 1 (*cf.*, 2.9), depending on the type of message encoding, without changing the plaintext [158].

BY ADDITION

$$\begin{array}{lll} {\scriptstyle r_1}\ m & g^{r_1}\ pk^{r_1}\ g^{m} & \\ {\scriptstyle r_2}\ 0 & g^{r_2}\ pk^{r_2}\ g^{0} & \\ {\scriptstyle r_1}\ m \quad {\scriptstyle r_2}\ 0 & g^{r_1}\ g^{r_2}\ pk^{r_1}\ g^{m}\ pk^{r_2}\ g^{0} & g^{r_1\ r_2}\ pk^{r_1\ r_2}\ g^{m} \end{array} \tag{2.8}$$

BY MULTIPLICATION

$$\begin{array}{lll} {\scriptstyle r_1}\ m & g^{r_1}\ pk^{r_1}\ m & \\ {\scriptstyle r_2}\ 1 & g^{r_2}\ pk^{r_2}\ 1 & \\ {\scriptstyle r_1}\ m \quad E_{r_2}\ 1 & g^{r_1}\ g^{r_2}\ pk^{r_1}\ m\ pk^{r_2}\ 1 & g^{r_1\ r_2}\ pk^{r_1\ r_2}\ m \end{array} \tag{2.9}$$

2.8 Mixnets

First introduced by Chaum in 1981 [51], the notion of mixnets originally addressed anonymous communication channels and untraceable electronic mail. Mixnets are essential building blocks for REV systems that preserve the privacy of voters. In short, mixnets irreversibly shuffle the encrypted ballots, which makes it computationally infeasible to link the encrypted ballots' ciphertexts to the decrypted plaintexts. Mixnets change the structure and order of ciphertexts such that the inputs to the mixnet can not be trivially linked to its outputs. It is crucial that the mixnet maintain the integrity of the ballots. It must not be possible that the mixnet adds or removes ballots or changes the content of a ballot. For that reason, a proof of a correct shuffle is required. This can be done using both Re-Encryption Mixnets (RMN) or Decryption Mixnets (DMN) [135, 158] (*cf.*, Fig. 2.6).

2.8.1 Decryption Mixnets

Decryption mixnets (DMN) were first introduced by Chaum [51]. A DMN works by using several layers of encryption. For each authority (mixer), the vote is encrypted with its public key. During every step of the mixnet, the ballots are stripped of another layer of encryption, resulting in a different shape and output. The results of the intermediate steps are still ciphertexts and, therefore, do not reveal any information about the votes. Only the sequential decryption of all encryption can reveal

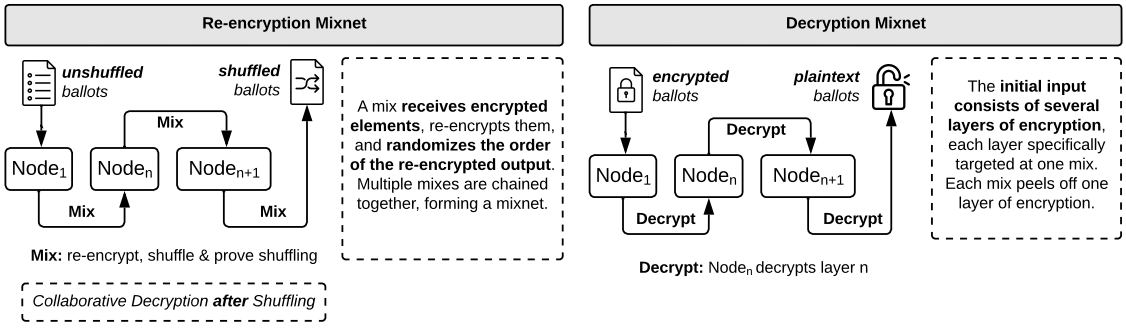


Figure 2.6: Types of Mixnets

the plaintexts and, in turn, the result of the vote. The approach is similar to the technique used in anonymous routing, such as in a TOR network [135, 158, 159].

2.8.2 Re-Encryption Mixnets

Re-encryption mixnets (RMN) were first introduced by Park et al. [239]. They require a probabilistic public key encryption system such as ElGamal [95] that allow for re-encrypting a ciphertext The process of a RMN can be compared with the physical task of shaking a ballot box to shuffle the ballots, therefore, making it impossible to associate a ballot with a specific voter [126, 130]. More formally, a RMN is formed by a series of cryptographic shuffles, each composed of a secret permutation and a re-encryption function. The cryptographic shuffle re-encrypts and permutes a group of ballots, *i.e.*, changes the ciphertext appearance and outputs the result in a randomized order. To avoid recoverability, this step can be repeated multiple times. The algorithms used in this thesis are based on the CHVote specification [126].

Definition 2.8.1. *(Permutation) A list of messages M* m_1 m_2 m_n *is considered permuted if the positions of all its entries are randomly reassigned M* m_5 m_n m_1 m_2 *while keeping the same number of elements. Also, none of the elements are allowed to change. A permutation is defined as a mapping* ψ 1 N 1 N *and can be selected at random from the set of all possible permutations using the Knuth (Fisher-Yates) algorithm [126, 130]. A pseudo-code version of the algorithm is illustrated in 2.7.*

Definition 2.8.2. *(Cryptographic Shuffle) A cryptographic shuffle transforms a list of ciphertexts* $\mathbf{c}$ c_1 c_2 c_n *into* $\mathbf{c}$ c_n c_2 c_1 *using permutation and re-encryption such that* c_j *re-enc* c_i *pk* j ψ i *[126, 130]. Pseudo-code of the algorithm is depicted in Fig. 2.8.*

GeneratePermutation

known: p q g
input: permutation size N
I 1 N
for i o N 1 **do**
r $_r$ i N 1
j_i $_1$ I r
I r I i
output: ψ j_1 j_N

Figure 2.7: Pseudo-Code Permutation Algorithm

GenerateShuffle

known: p q g
inputs:
encryptions **c** c_1 c_N
public key pk
ψ j_1 j_N $_r$ Ψ_N (see 2.7)
for i 1 N **do**
j_i ψ i
c_i `re-encrypt` c_{j_i} pk
output: c c_1 c_N

Figure 2.8: Pseudo-Code Shuffle Algorithm

In summary, a multiplicative HE c E_{r_1} m pk can be re-encrypted as c re-enc c pk E_{r_2} c pk c E_{r_2} o pk . The output c represents the same plaintexts (m_1 m_2 m_n) as c but their ciphertext representation is different and in permuted order [126, 130]. The correctness of the cryptographic shuffle needs to be proven for each re-encrypted message. This is called a *proof of shuffle* and is explained in detail in Section 2.9.5.

In both DNM and RNM, the outcome is privacy-preserving, as an adversary cannot identify voters using their ballots. Nevertheless, RMNs have become the de-facto standard for REV systems leveraging mixnet techniques. This is because mixnets in REV systems come into play once all ballots have been cast and the voters' signatures have been verified. RMNs are advantageous in this case because the voters only need to perform a single encryption while the shuffling process can be repeated multiple times independently of the voters' original encryption [126, 130, 135, 142, 158, 159].

2.9 Zero-Knowledge Proofs

Zero-Knowledge Proofs ( ) are a powerful cryptographic mechanism allowing a prover to prove knowledge about a secret *s* to a verifier without revealing anything about *s*. can be used to prove Boolean statements over arguments, which are either private or publicly known, while not revealing any private arguments. In REV systems, are used in many different ways, *e.g.*, to prove knowledge of a private key belonging to a specific public key [158].

must satisfy the following three properties [115]:

1. **Completeness**: if the statement is true, the honest will accept it.
2. **Soundness**: if the statement is false, it is not possible to convince an honest that the statement is true.
3. **Zero-Knowledge**: no malicious can learn anything about the secret from the proof, except for its correctness.

Interactive are formulated as a challenge-response game in which first commits to a statement, and secondly, the sends a challenge to the , to which responds. These three-move protocols are referred to as Σ-protocols [68, 115](*cf.*, Fig. 2.9). The repeated execution of this commit-challenge-response scheme enables to gain confidence in the proof [158] gradually.

provide probabilistic soundness guarantees, *i.e.*, the probability that a malicious prover can craft a valid proof for a false statement that an honest verifier will accept, is negligible but non-zero. In REV systems, can be used to prove that a ballot is valid and well-formed without leaking anything about its contents.

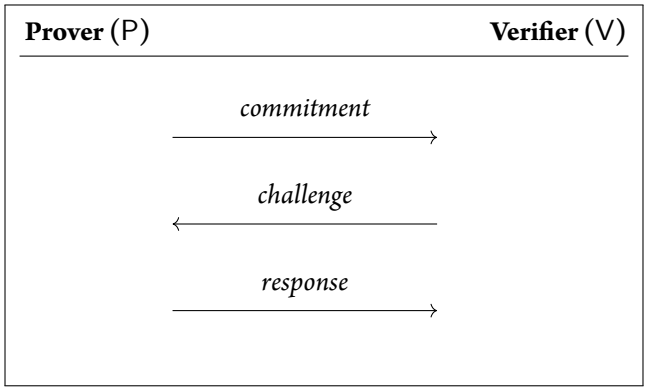


Figure 2.9: Three-Move Σ-Protocol

2.9.1 Non-Interactive ( )

In theory, Σ-protocols are sufficient to prove knowledge of a statement. However, in real-world applications, the communication overhead of interactive protocols is not practical and do not see any use in applied cryptography. Fortunately, if a statement can be proven with a Σ-protocol, it can be turned into a non-interactive proof by replacing 's challenge with an unpredictable hash digest of the *commitment and the statement* [278]. do not require interaction and can be generated by alone, while the same goes for the verification by , which does not require .

The *Fiat-Shamir Heuristic (FS)* [100] is the most common technique to turn interactive Σ-protocols *non-interactive* by replacing the random challenge generated by the with a hash of the algorithm's transcript, which then serves as the random oracle, *i.e.*, the source of uniform randomness. This allows to be easily generated and published, allowing anyone to verify a statement without information leakage.

Different protocols have used the FS heuristic to reduce communication overhead in REV schemes. [27] identified that two distinct applications of such heuristics exist in the literature: a strong variant (sFS) and a weak one (wFS). While sFS hashes the commitments and the statement, wFS *only* hashes the commitments. This can lead to unsound proofs in the case of malicious provers: *i.e.*, proofs that do not actually prove the statement that they are supposed to prove.

are a central tool used widely in REV protocols to reduce trust assumptions and provide verifiable proof without leaking any information about the data at hand. For instance, in schemes based on homomorphic encryption (HE), ballot validity are used to prove that a ballot is formed correctly and does indeed contain a valid vote (for instance, a 0 or 1 vote). Also, are used to prove the correct decryption of shared tallies, for instance, when multiple entities own a private part of the election key, which can be implemented using Secret Sharing techniques (*cf.*, Section 2.5).

2.9.2 Schnorr Identification Algorithm

One of the simplest and most commonly used proofs of knowledge is generated with the Schnorr Identification Algorithm. Schnorr Proofs can be used to prove knowledge of an ElGamal private key , belonging to a public key (*i.e.*, g^{sk}) [269]. Schnorr Proofs are a proof of knowledge of a discrete logarithm of $log_g\ g^{sk}$ [97]. Since REV systems frequently use the ElGamal public key cryptosystem (*cf.*, Section 2.3.1), Schnorr Proofs can be used to prove knowledge of a secret key share of the election key.

Equation 2.10 depicts the Schnorr Proof generation. The secret key ( ), as well as the public key ( ), are required as inputs. Additionally, a secure random value of a in the range 1 q 1 is needed. q $\frac{p-1}{2}$ is the additive modulus of the cyclic group Z_q over the field F_q (*i.e.*, the cyclic group in which all additive operations take place) and g is the group's generator. The id is the unique identifier of the prover. First, the commitment b, the generator g to the power of a, are hashed together with the unique id and the public key . This forms the challenge c that makes up the first part of the proof. The second part of the proof is the response, which is the sum of the random value a and the product of the challenge c and the private key .

(2.10)

input	keypair
commitment	$b \quad g^a \mod p$
challenge	$c \quad hash\ id \quad b \mod q$
response	$d \quad a \quad c \quad \mod q$
output	proof $c\ d$

(2.11)

In order to verify a Schnorr proof $c\ d$, first, the commitment b and challenge c need to be recomputed (*cf.*, Equation 2.12). Then, the recomputed challenge c needs to match the original challenge c. Further, the generator g to the power of the response d needs to be equal to the recomputed commitment b multiplied by the public key ( ) to the power of the original challenge c. If both checks are successful, the proof verification concludes.

(2.12)

input	proof $c\ d$
commitment	$b \quad g^d\ h^c \mod p$
challenge	$c \quad id \quad b \mod q$
compare hashes	$c \quad c$
verify proof	$g^d \mod p \quad b \quad ^c \mod p$

(2.13)

Finally, in Figure 2.10, the interactions between and are outlined. With the following protocol, can prove knowledge of to .

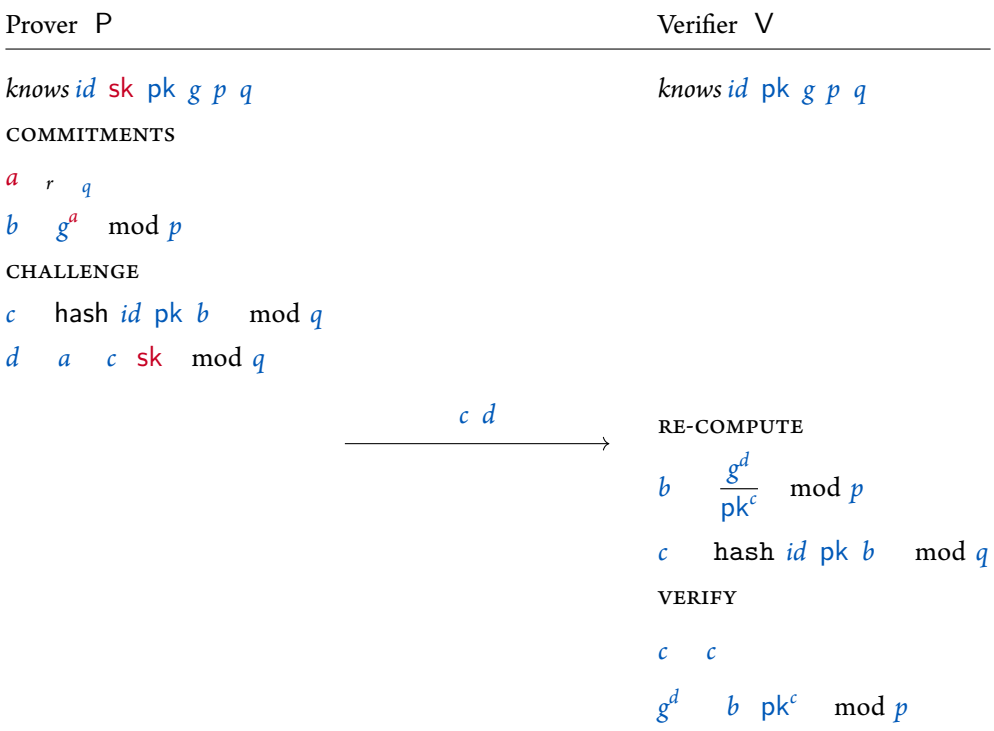


Figure 2.10: Schnorr Proof for Key Generation

2.9.3 Chaum-Pedersen Protocol

The Chaum-Pedersen protocol [50] is used to generate a proof of discrete logarithm equality. The protocol can be used to prove correct re-encryption of ciphertexts, without revealing the secret randomization factor in ElGamal. Moreover, the protocol can be used to prove correct decryption of a ciphertext.

To generate the proof for a ciphertext c_1 c_2 (*cf.*, Equation 2.14), a secret random value x in the range 1 q 1 is required. The commitment consists of a and b, whereas a is equal to the cipher's first component c_1 to the power of x, and b equals the generator g to the power of x. Next, the challenge c is computed by hashing the unique id of the prover, the cipher's components c_1 and c_2, and components of the commitment a and b with a cryptographic hash function. The final response consists of f and d. f consists of the addition of x to the product of the challenge c and the private key . Also, d consists of the cipher's first component c_1 to the power of the private key . The final proof consists of the commitment (a, b) and the response (f, d).

(2.14)

input	ciphertext	c_1 c_2
commitment	a	$c_1^{x} \mod p$
	b	$g^{c} \mod p$
challenge	c	$hash\ id\ c_1\ c_2\ a\ b \mod q$
response	f	$x \quad c \mod q$
	d	$c_1^{sk} \mod p$
output	proof	$a\ b\ f\ d$

The the Chaum-Pedersen protocol's verification algorithm consists of the commitment (a, b) and the response (f, d) for a ciphertext (c_1, c_2) (*cf.*, Equation 2.15). The challenge c is recomputed by hashing the prover's unique id together with the cipher (c_1, c_2) and the response (a, b) extracted from the proof. To verify the proof's validity, two checks are performed: *(i)* the cipher's first component c_1 to the power of the response's fist component f needs to be equal to the commitment's first component a multiplied by the response's second component d to the power of the recomputed challenge c and *(ii)* it is checked, whether the generator g to the power of the response's fist component f is equal to the commitment's second component b multiplied with the public key to the power of the recomputed challenge c .

(2.15)

input	proof	$a\ b\ f\ d$
	ciphertext	$c_1\ c_2$
challenge	c	hash $id\ c_1\ c_2\ a\ b \mod q$
verify proof	$c_1^{f} \mod p$	$a \quad d^{c} \mod p$
	$g^{f} \mod p$	$b \quad {}^{c} \mod p$

To prove that a partial decryption of a ciphertext c a b is correct, the following equality needs to be proven: d a^{sk_i}, where pk_i g^{sk_i}. The proof used in this work (2.11) is based on the CHVote specification [126]. The decryption proof is a modified version of the Chaum-Pedersen proof [50]

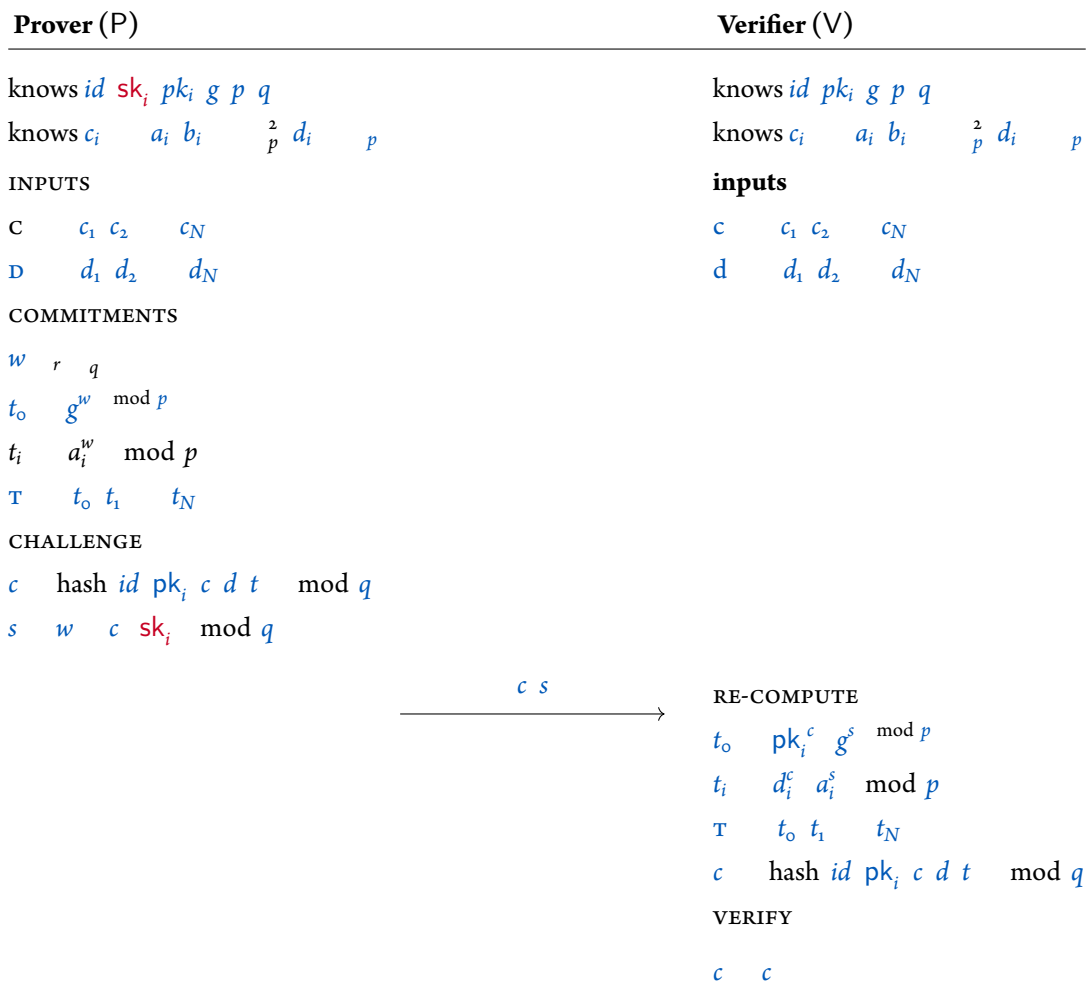


Figure 2.11: Decryption Proof

such that a single proof can show the correct partial decryption for a batch of ciphertexts. It is a proof of discrete logarithm equality of $\log_g g^r$ $\log_h h^r$, and comparable to the Schnorr proof [269].

Disjunctive Chaum-Pedersen Proof

The disjunctive Chaum-Pedersen proof is a modification of the Chaum-Pedersen protocol [221]. The disjunctive variant is also called OR-proof or membership proof. Applying this to ballots in the context of voting systems, disjunctive proofs prove that the vote contained in the ballot is proven to be either a `no` vote (value 0, subsequently represented as v_0) or a `yes` vote (value 1, subsequently represented as v_1) and guaranteed to contain nothing else. The proof does not leak anything about the ballot's actual content, which makes it feasible to prove valid ballot encryption (*e.g.*, in voting systems based on HE). The proof generation protocol consists of two parts: one part for v_0 and one

for v_1. For instance, if the encrypted ballot contains a `yes` vote (*i.e.*, v_1), the `no` vote of the proof needs to be simulated, and vice versa. In Equation 2.16 a `yes` vote proof is depicted, whereas c_0 f_0 x are secure random values in the range 1 q 1 and is the random value during vote encryption.

(2.16)

input	ciphertext	a b
simulate v_0	a_0 b_0	g^{f_0} a^{c_0} mod p ^{f_0} b^{c_0} mod p
proof v_1	a_1 b_1	g^{x} mod p ^{x} mod p
challenge	c	*hash* *id* a b a_0 b_0 a_1 b_1 mod q
	c_1 c c_0	mod q
response	f_1 x c_1	mod q
output	v_1 proof	a_0 b_0 a_1 b_1 c_0 c_1 f_0 f_1

If the encrypted ballot contains a `no` vote instead of simulating c_0 f_0; the values c_1 f_1 (secure random values in the range 1 q 1) are simulated instead. Another slight change is that instead of using b in the simulation, b g is used. Actually, in the v_1 proof, b g^0 is taken, which equals b. In the v_0 proof, where the `yes`-vote (of value 1) is simulated, b g^1 is taken [221].

(2.17)

input	ciphertext	a b
simulate v_1	a_1 b_1	g^{f_1} a^{c_1} mod p
		^{f_1} b g ^{c_1} mod p
proof v_0	a_0 b_0	g^{x} mod p ^{x} mod p
challenge	c	*hash* *id* a b a_0 b_0 a_1 b_1 mod q
	c_0 c c_1	mod q
response	f_0 x c_0	mod q
output	v_0 proof	a_0 b_0 a_1 b_1 c_0 c_1 f_0 f_1

Both proofs for v_1 and v_0 are verified with the algorithm depicted in Equation 2.18. The ciphertext serves as input *(i)*, while the recomputed challenge *(ii)* is followed by a comparison of hashes *(iii)* and concludes with various checks executed to validate the proof *(iv)*.

(2.18)

(i) ciphertext $a\ b$

proof $a_0\ b_0\ a_1\ b_1\ c_0\ c_1\ f_0\ f_1$

(ii) c *hash id* $a\ b\ a_0\ b_0\ a_1\ b_1 \mod q$

(iii) $c_0\ c_1 \mod q\ c$

(iv) $g^{f_0} \mod p\ a_0\ a^{c_0} \mod p$

$g^{f_1} \mod p\ a_1\ a^{c_1} \mod p$

$^{f_0} \mod p\ b_0\ b^{c_0} \mod p$

$^{f_1} \mod p\ b_1\ b\ g^{\ c_1} \mod p$

2.9.4 Designated Verifier Proof

Regular are transferable, meaning that the proof can be verified by anyone, not just . Informally, a Designated Verifier ( ) should convince the of the correctness of the statement since the *knows* that he did not generate the proof himself. However, no other party will be convinced of the proof's validity since the *could* have created it. For example, if a voting system used a regular , a voter would be able to convince with other voters' proofs by simply forwarding the proof. To mitigate this risk, a can be transformed to be non-transferable by adapting it so that it only convinces a . A proof proves a statement similar to:

knowledge of 's OR knowledge of the secret

Since only the has knowledge of the , the is the only one able to verify the claim's correctness [158, 281].

In this thesis, a proof is used by the randomizer component to re-encrypt and then prove to a voter that an encrypted ballot c is a valid re-encryption of the voter's original encrypted ballot c. The main concept is to re-encrypt with the identity element, *i.e.*, 0 (addition) and 1 (multiplication), to change the appearance of the resulting ciphertext but not the content of the ballot. The re-encryption variants can be seen in Section 2.7 in Equations 2.8 and 2.9. The proof protocol shown in 2.12 is inspired by Hirt *et al.* [146] and makes use of the Schnorr proof [269]. It has been adapted for use with non-encoded, multiplicative homomorphic encryptions (*cf.*, Section 2.3.1). The proof is a

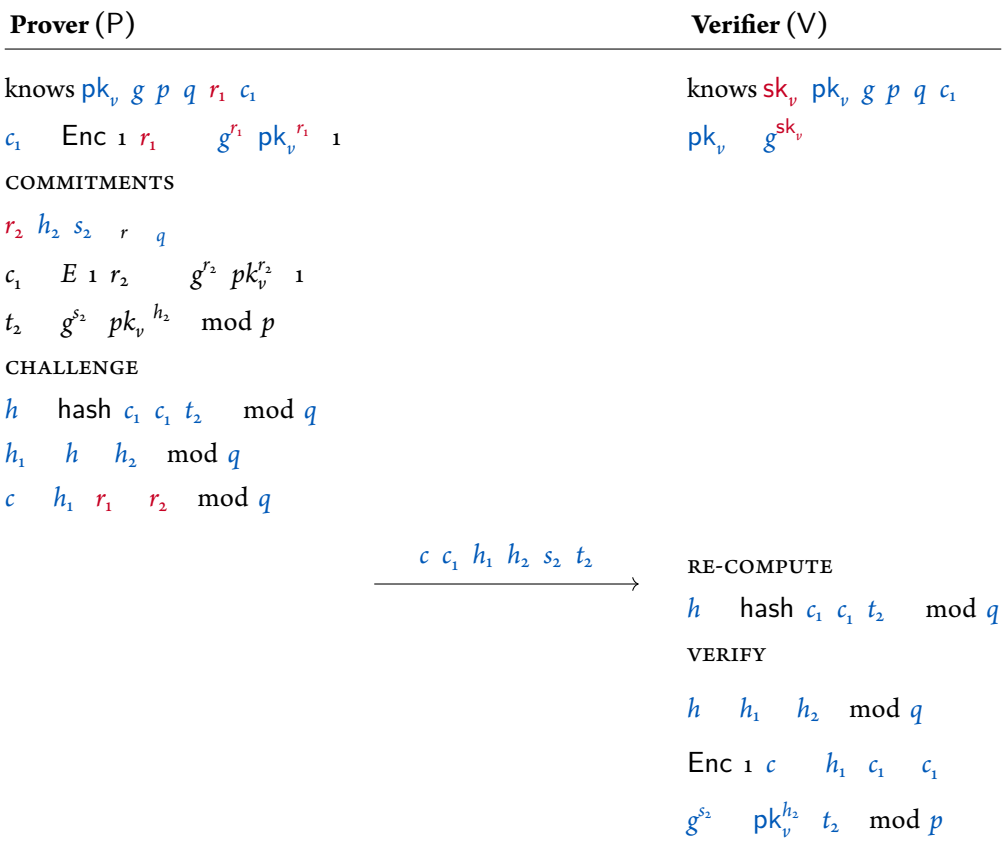


Figure 2.12: Designated Verifier Re-Encryption Proof

logical OR-combination of two parts, a proof of knowledge of a discrete logarithm such that $_{voter}$ $\log_g$ $_{voter}$ $\log_g g^{sk_{voter}}$. And, a proof such that c_1 c c c c ^{1} is satisfied, given c m r_o (the voter's initial encryption), c_1 1 r_1 (an encryption of 1 using the re-encryption random value r_1) and c re-encrypt c r_1 c_1 c (the randomizer's re-encryption of c). Referring back to the statement to be proven, the *secret* is the random value used in the re-encryption r_1.

2.9.5 Shuffle Proof

Shuffle proofs are that can be used to prove the correct shuffling of ciphertexts (*e.g.*, shuffling ballots in a mixnet-based REV system) and therefore are sophisticated and powerful cryptographic tools. The shuffle proof used in this work was first introduced by Furukawa and Sako [106] and is based on permutation matrices. The exact implementation is a proof of a correct shuffle [142, 304] and adapted from the CHVote specification [126]. In the following, the necessary concepts are introduced to understand how the proof works and what it is composed of. The auxiliary algorithms ref-

erenced in both the generation and verification are described in Appendix B, depicted in Figures B.3, B.4, B.5 and B.6.

As briefly introduced in Section 7.4.1, the correctness of a cryptographic shuffle, which transforms a list of ciphertexts c c_1 c_2 c_n into c' c_n c_2 c_1 using permutation and re-encryption such that c_j re-encrypt c_i r_i j ψ i , needs to be proven for each re-encryption. This can be done by proving knowledge of the permutation ψ and the vector of random factors used for re-encryption r r_1 r_2 r_n [126, 130]. The proof looks as follows:

$$\pi_{\psi} \quad \psi \; R \quad c_j \quad c_i \; r_i \; \psi \quad j \quad \psi_i \; i \tag{2.19}$$

The result is a triple of c c π_{ψ} containing the initial list of ciphertexts c, its shuffled (*i.e.*, re-encrypted and permuted) transformation c and the proof attesting to the validity of the shuffle. Unfortunately, standard pre-image proofs such as the Schnorr proof (*cf.*, Section 2.9.2), cannot be applied to the cryptographic shuffle as it is not homomorphic, which is a requirement to make the proof non-interactive [130]. To render such a feasible for usage in a REV system, it needs to be possible to transform it into a for practical reasons. Therefore, an alternative homomorphic version of the proof is required. Shuffle proofs are actively researched, with many different approaches and strategies being explored. One of the most widely explored alternative proofs was proposed by Wikström and Terelius [304]. It is the basis for the protocol of this work and composed of an *online* and *offline* partm which are outlined as follows.

Definition 2.9.1. *(Pedersen Commitment) A protocol that allows a prover to commit to a secret value m without revealing it or being able to change it later on: c m r $g^m h^r$ mod p. h and g are independent generators of G_q, the same subgroup of Z_p (p $2q$ 1) that the ElGamal encryption system uses. The commitment c can be stored publicly without revealing any information about the secret value m. Once reveals m and r, any verifier can check that c m r $g^m h^r$ mod p.*

In the first part (which is offline), the computes a Pedersen commitment to a permutation matrix: c ψ r , where the value of the commitment is the permutation function ψ [130, 304].

Definition 2.9.2. *(Permutation Matrix) A permutation matrix B_{ψ} b_{ij} N N fulfills the following two conditions: (1) the elements of each row must sum up to one: $\sum_{j=1}^{N} b_{ij}$ 1 and (2) each row contains exactly one non-zero element: $\sum_{i=1}^{N} \sum_{j=1}^{N} b_{ij} x_i$ $\sum_{i=1}^{N} x_i$. Therefore, the permutation matrix is composed of rows with a single one while all other entries are zeros [130].*

In the second part (which is online), shows that the permutation matrix is used in the cryptographic shuffle to transform c to c using the standard pre-image proof. Now, one must show in

zero-knowledge that the two conditions introduced in definition 2.9.2 are satisfied for B_{ψ}. Together, with the commitment from the offline part and the proof for correct re-encryption, one has created a zero-knowledge proof for a cryptographic shuffle without revealing any information about the permutation ψ or the re-encryption random values r [130, 304].

We need to realize popular services in a secure, distributed, and decentralized way, powered by free software and free/open hardware. We need to build systems beyond the reach of super-sized companies and spy agencies. Such services must be based on strong cryptography.

Phillip Rogaway [257], (2015)

3 Voting System Fundamentals

This chapter outlines the general requirements and building blocks needed to build a voting system that uses cryptographic primitives successfully. Furthermore, as this thesis focuses on *remote* voting systems, the following sections also emphasize aspects of *remote* systems, although many of the requirements also apply to *on-site* voting systems. After that, the detailed requirements for decentralized voting systems are outlined. Finally, the different voting scheme approaches are described in detail.

3.1 Remote Electronic Voting Requirements

The core challenge in Remote Electronic Voting (REV) is to balance privacy and verifiability properties using a transparent and evidence-based voting process. Many early REV systems used trusted central servers as a Public Bulletin Board (PBB), representing a single point of failure operated by trusted stakeholders that may be interested in the election's outcome [132]. For example, in Switzerland, the Swiss Post (SP) Voting system operates in trusted, closed, and hardened data centers under the control of the SP (a partially state-owned company). Although the protocol assumes only a set of separate control components as trustworthy, they are all technically operated by the SP [134].

The overall system also depends on a code-verification system, which assumes a trustworthy printing office that secretly prints and sends code sheets via postal mail to eligible voters. In the case of collusion between the printing office and compromised devices, the device can return correct verification codes, even though the votes are not cast as intended by the voter [23].

Ideally, a cryptographic voting scheme would require voters only to trust cryptographic schemes. Assuming external stakeholders' trustworthiness (*e.g.*, a print office's printing verification code) would not be required. To minimize these trust assumptions placed in centralized infrastructure, Blockchains (BC) and Distributed Ledgers (DL) provide a suitable possibility to *(i)* minimize and distribute trust among stakeholders and *(ii)* enable practical implementation of an immutable and append-only PBB for REV systems. However, many of the BC- and DL-based systems proposed did not consider the complete set of requirements for REV systems, *e.g.*, a trust model and assumptions of underlying consensus mechanisms. Persisting election evidence on a BC or DL enhances trust assumptions, mainly because the election evidence's immutability is assured. Further, the practical decentralization of the voting process on an architectural layer can be achieved as well.

The design goals are similar and reasonable: increase transparency and provide a publicly readable and fully decentralized audit trail. After all, transparency is a critical factor for trust in REV systems. For instance, the Swiss Federal Ordinance for Federal Chancellery Ordinance on Electronic Voting (VEleS) [84] demands the publication of source code for any REV system used to tally more than 30% of the cantonal electorate. To assess the feasibility of any system, the protocol and system requirements, the architecture, and its components need to be evaluated. Therefore, the primary building blocks of a REV system are essential, with the core component of a decentralized REV system being built by the Public Bulletin Board (PBB). Voters use a Voting Client that executes a Voting Software Client (VSC) to interact with the PBB directly through a set of defined communication channels while interacting according to the voting protocol to ensure privacy and verifiability (*cf.*, Fig. 3.1.1).

3.1.1 Protocol Requirements

Literature on EV offers various protocols to meet the core requirements of privacy and verifiability, which are in direct opposition to one another and inherently contradictory(*cf.*, Fig. 3.1), *e.g.*, Individual Verifiability (IV) seems to be incompatible with Receipt-Freeness (RF) [125] . Also, [52] has shown that a voting system cannot achieve Unconditional Privacy (UP) and Universal Verifiability (UV) at the same time. They also showed that UV and RF could not coexist if the transcript of the

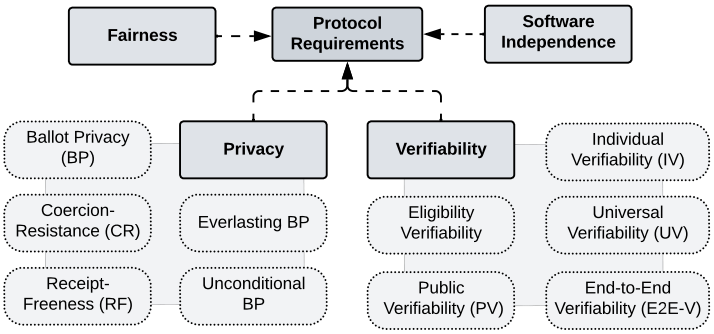


Figure 3.1: Overview of Protocol Requirements

vote depends on *only* public values and values known to the voter. For this reason, REV systems aim to achieve a balance between privacy and verifiability.

Privacy Properties

The privacy of a vote is considered a human right and is stated as such in Article 21 of the Universal Declaration of Human Rights [306]. A voting system without ballot privacy allows large-scale vote-buying and voter coercion. Thus, privacy is key in democratic voting. In this context, privacy can be further classified in subcategories, which are based on the degrees of privacy and assumptions made, ranging from the weakest notion of Ballot Privacy (BP) to stronger notions, such as Coercion Resistance (CR) [73].

Many privacy properties require at least a minimal amount of trust placed in the Voting Authorities (VA) to store the private keys for the election keys safely. Trust in single authorities can be distributed by using threshold cryptography, hence distributing the trust among VAs and other trusted stakeholders. For instance, by using a BC or DL-based system, it is possible to decentralize the point of trust in a single party through transparent infrastructure in which all participants can verify the distributed key generation, the confirmation of votes cast, and the vote tallying procedures.

> The will of the people shall be the basis of the authority of government; this will shall be expressed in periodic and genuine elections which shall be by universal and equal suffrage and shall be held by secret vote or by equivalent free voting procedures.
>
> Article 21, (3) of the Universal Declaration of Human Rights [306], 1948

In 1981, the first notion of *Ballot Privacy (BP)* was defined by [48] as "A voter's vote is not revealed to anyone". In [55], the notion was formally introduced, laying the groundwork for formal notions of

voting system properties. Often, the term *Ballot Secrecy* is used interchangeably to describe BP. The National Institute of Standards and Technology (NIST) defined Ballot Secrecy in a more process-oriented fashion as follows: "a voting system is private if it (and the procedures/process for using it) does not make available additional information on an individual voter's ballot choice(s), beyond that contained in the tally" [208]. Later, in 2002, another notable definition followed from [171], called *Perfect Ballot Secrecy*, which "ensures that knowledge about the partial tally of the ballots of any set of voters is only computable by the coalition of all the remaining voters (this property captures strong voter privacy as understood in real world elections)". Further refinements of this definition can be found in literature based on the trust model or the computational power of an adversary [171, 203, 225].

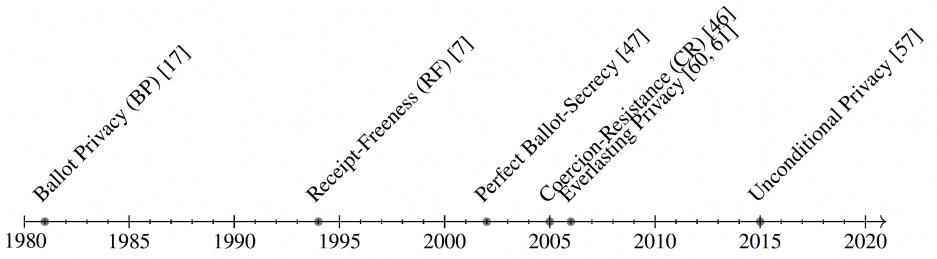


Figure 3.2: Timeline of Privacy Notions

Most early works on voting systems had underlying protocols that produced receipts. These receipts would allow voters to prove how they voted to others. However, the ability to produce a receipt of how a voter voted can easily be abused for vote-buying or coercion [25]. This brought about the definition of *Receipt-Freeness (RF)*, which is defined as "a voter cannot gain information which can be used to prove, to a vote-buyer, how she voted" [25].

The key difference between RF and *Coercion-Resistance (CR)* is whether the voter is considered malicious or not. In the case of RF, the voter is considered to be a dishonest voter who wishes to sell her vote. For CR, it does not matter whether the voter is honest or not. The voter is coerced by an adversary to disclose sensitive information, such as her vote or voting credentials. As such, CR is a stronger privacy guarantee than RF. Therefore, CR is defined as "a voter cannot interact with a coercer to gain information, which can be used to prove how she voted" [158].

The privacy definitions above do not explicitly define an adversarial model *i.e.*, do not include any assumptions regarding the adversary's computational capabilities. While current adversaries are bound by hardware and software available at the moment, future adversaries may break underlying assumptions, which in turn break the security of cryptographic primitives and compromise the privacy

properties. Many voting schemes are based on these hardness assumptions, such as the Decisional Diffie-Hellman assumption (DDH) [30], which is covered in Section 2.1. With a focus on these intractability assumptions, further definitions of *everlasting* and *unconditional* BP notions are reviewed.

Per definition, a voting system with *Everlasting BP (EP)* does not allow even a computationally unbounded coalition of voters or outside observers to determine for whom a voter voted [225]. Going even further is *Unconditional BP (UP)*. A voting system with UP does not allow anyone, neither voters, outside observers, nor VAs, be they computationally unbounded or not, to determine for whom another voter voted [203].

Remembering the definition of perfect BP [171], even if VAs conspire, they will still be unable to link votes to voters and break BP. Therefore, a voting scheme with perfect BP does not require trust in the VAs or other parties. However, perfect BP does not mention any computational hardness assumptions and is not everlasting. Therefore, by combining everlasting BP (*i.e.*, no computational hardness assumptions for privacy) and perfect BP (*i.e.*, no trust in TTPs needed for BP), it is possible to achieve Unconditional BP.

Verifiability

In real-world systems, voters have to place implicit trust in voting machines and polling staff for the integrity of the elections. In the pursuit to minimize the requirement for trusted personnel and hardware, REV research has proposed cryptographically verifiable voting schemes [5].

Once all votes have been cast, the system must be able to prove that the result was tallied correctly and that the single ballots contain the actual voters' choices. Prior work proposed many different notions of verifiability, and novel, formal definitions are being researched as well. [263] defined *Individual Verifiability (IV)* as "a voter can verify that her vote is included in the set of all votes cast".

However, IV is not sufficient to provide a verifiable REV system because the system then relies on each voter to verify that each ballot was published correctly and unaltered. Since that is an unrealistic assumption, [263] also defined Universal Verifiability (UV) as being achieved if "an observer can verify that the tally has been correctly computed from the set of all votes cast" [263].

Additional informal definitions of UV by Kremer *et al.* [185] are defined as being achieved if "anyone can verify that any vote in the set of 'all' votes belongs to an eligible voter, and each eligible voter voted only once [185]". This notion already includes eligibility verifiability, highlighting the challenges of ensuring that only eligible voters vote and only vote once.

Further research in the realm of verifiability properties refined a more procedural perspective on the *end-to-end* voting process, called *End-to-End Verifiability (E2E-V)*, which is only fulfilled if a voter can verify the following three properties [158]:

- *Cast-as-Intended* (CaI): "her choice was correctly denoted on the ballot by the system".
- *Recorded-as-Cast* (RaC): "her ballot was received the way she cast it".
- *Counted-as-Recorded* (CaR): "her ballot counts as received".

The last property of E2E-V, CaR, is often also referred to as *Tallied-as-Recorded* (TaR) as well. Therefore, voters first confirm to the system that the vote has been correctly encrypted, and thus CaI is achieved (by verifying the encryption operation on a second device, for instance). Then, the voter tracks her vote on the by using her receipt and can verify that the vote was RaC. Lastly, the result's integrity is ensured by auditing the tallying process, which means that the REV system publishes cryptographic proofs that prove correct decryption and operation [5].

E2E-V and the combination of IV and UV cover similar aspects. In research, usually either E2E-V or the combination of IV and UV is applied, but not both simultaneously. IV and Cast-as-Intended verifiability (CaIV) address the problem of a non-trusted voter platform in REV. CaIV needs to be addressed if the voter's platform cannot be controlled and trusted, which is typically the case when the voter is able to cast an electronic ballot remotely. IV and CaIV together assure that the voter can detect that her vote was compromised, most probably by malicious code on the voting device [175]. While IV, UV, and E2E-V are of central importance for REV schemes, they are hard to prove in combination, but even harder to be deployed within an operational and practical RPV system. Therefore, the following notions of verifiability are necessary when existing systems are evaluated, such as the Swiss RPV system.

Eligibility Verifiability can be defined as *A voting system has the eligibility verifiability property if anyone can verify that the voting result only contains votes from eligible voters, and only includes one vote per voter* [185]. Eligibility verifiability is crucial for any voting system, independent of being based on paper ballots or operated electronically, since it ensures the integrity of the final tally, making it verifiable that only eligible voters cast their vote once.

Further, Administrative Verifiability (AV) is defined as *A voting system offers AV when election officials have means to protect against certain kinds of errors and fraud, typically accomplished with tools like paper audit trails that enable manual recounts and spot checks* [21].

Lastly, Public Verifiability (PV) is defined as textitA voting system provides PV when any individual can verify the accuracy of a tally regardless of any conspiracies of any size [21].

Based on these definitions, AV includes various processes implemented by trusted authorities, while PV requires cryptographic mechanisms to be deployed. In other words, AV determines real-world processes in operation and PV is based on an ideal, theoretical property and is hard to deploy in practise.

However, these verifiability properties rely on a as an audit trail. Otherwise, there is no data to verify. A serves as an immutable, append-only audit trail. Therefore, verifiability properties *assume* such a exists. In practice, a fully decentralized REV system requires a shared . A public permissioned DL can serve as such and enable these different verifiability properties, allowing voters, auditors, and the general public to query the persisted data. It is crucial that the deployment of said BC or DL be carefully parameterized and configured to the use-case and voting protocol. For instance, there is a huge difference between simply persisting hashes on a public permissionless BC, and having actual VAs perform Multi-Party Computation within a permissioned DL network. These differences make it necessary to look at the properties within the specific context and mode of deployment of the .

[200] drove efforts forward to elevate the understanding of dependencies between various privacy and verifiability properties in their formal and informal definitions, as well as misconceptions and misunderstandings of these properties.

1. For instance, it is commonly believed that IV and UV are the same as E2E-V. However, IV and UV are neither sufficient nor necessary to achieve E2E-V.

2. Further, E2E-V alone is typically insufficient for practical purposes. REV systems should really be designed with accountability in mind.

3. Further, CR does not imply general privacy, but increasing the level of privacy can lead to a lower level of coercion-resistance.

These confusions and misconceptions highlight the importance of clear definitions and a clear assessment of these definitions. The privacy and verifiability notions are not fixed, but are ever-evolving into different formalized approaches. An overview is provided in [200]. Also, privacy and verifiability are not the *only* protocol requirements. Two notable properties include the notions of Software Independence and Accountability in the context of REV systems.

Additionally, the requirement *Software Independence* can be defined as *if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome* [255]. REVs are complex hardware/software systems. Minor errors or manipulations can lead to unpredictable errors, which adversaries might exploit. As such, Software Independence follows the approach of: *Verify the election results, not the voting system*.

Furthermore, *Accountability* is a more robust, extended form of verifiability, *i.e.*, if the final tally does not correspond to how the voters actually voted, then VAs can be held accountable for the misbehavior. The misbehavior can be intentional or unintentional. Accountability is important for mul-

tiple practical reasons because it intensifies the incentive all parties have to follow their roles, as they can be singled out in cases of misbehavior [200].

3.1.2 General Voting System Requirements

While these protocol requirements are essential for formulating voting protocols, the REV system also needs to provide multiple facets, many of which are not specific to REV systems but also to information systems in general. These general security system requirements are less specific to REV but still fundamental in practice, as REV follows a clear software architecture (*cf.*, Fig. 3.3). These general requirements may contradict protocol requirements [125]. For instance, software architecture characteristics usually refer to the *"-ilities"* that the system must provide: availability, reliability, testability, scalability, deployability [253]. Also besides these general requirements, convenience, flexibility, and usability are also desirable properties. Further requirements that address political, administrative, or even juridical questions are also crucial for REV in practice.

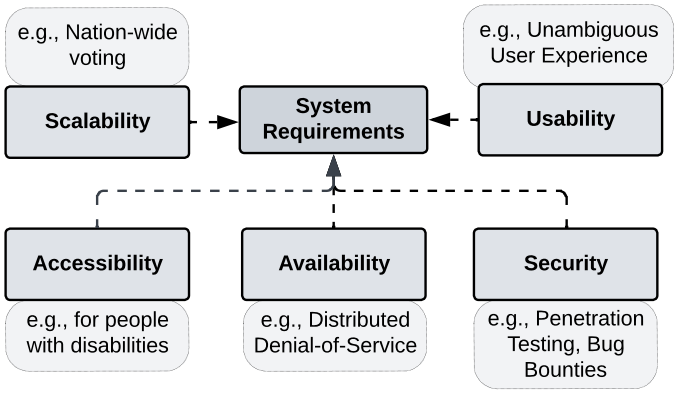


Figure 3.3: Overview of System Requirements

In the context of BC or DL-based REV systems, the most important aspect is defined security goals and the threat model because these build the basis for the system's risk assessment. For instance, when using a public permissionless BC as a basis, the system's architecture depends on the assumption that the public network is reliable, available, and scalable. Public permissionless networks are, by design, not controlled by known stakeholders. Therefore, it is hard to challenge or prove the properties and assumptions made. However, if a voting system relies on a known set of permissioned nodes, *i.e.*, a public permissioned DL, then the assumptions, properties, and performance can be measured more precisely while still providing the benefits of transparency.

3.2 From Centralized to Decentralized REV Systems

The primary goal of REV systems is to achieve a privacy-preserving and end-to-end verifiable voting process. The approaches outlined in Section 3.4 show various directions to reach an overall secure process. While the primary responsibility of the is to persist verifiable information immutably, the architectural model is not always considered carefully since the existence of such a is either explicitly or implicitly stated as an assumption. Also, depending on the voting protocol chosen, the trust model is not influenced substantially either. For instance, if a single TTP stores private keys for a given election, the integrity depends solely on that TTP, whether the is deployed in a distributed setting or not.

It is necessary to differentiate between a public permissionless BC and a public permissioned DL operating as a since the underlying consensus algorithm's assumptions differ and impact key assumptions about trust (*cf.*, Section 3.3.5). For example, a BC (*e.g.*, Bitcoin) follows a permissionless, or generally spoken, "trustless", and fully decentralized trust model, where it is not necessary to trust entities, only *(i)* their actions, which are publicly persisted on the BC and *(ii)* the cryptographic mechanisms in use. This implies a paradigm shift from placing trust directly in a centralized entity instead of trusting a decentralized platform's cryptographic mechanisms, which enables transparency and public verifiability. However, it is crucial to consider privacy aspects and ensuring that Ballot Privacy (BP) is maintained under all circumstances, which means that, *e.g.*, Bitcoin's fully public and permissionless approach becomes inadequate for guaranteeing scalable confidentiality.

Therefore, a generic centralized voting system's architecture must take these considerations into account. In the proposal below, the trust assumptions are analyzed, and a broader risk assessment is performed. This leads to a new, fully decentralized architecture proposal and a discussion of its trade-offs.

3.2.1 Centralized REV System

The architecture of a centralized REV system (*cf.*, Figure 3.4) is decoupled from a specific voting protocol. Major stakeholders include voters and one Voting Authority (VA). Voters are provided with the VA's voting credentials and are equipped with technical means to cast their ballots in their voting client. These credentials can be provided electronically (*e.g.*, via electronic mail) or physically (*e.g.*, via postal mail) by a VA for either mixing or decrypting the final tally of votes. In most cases, a voting client is a piece of software executed on voting clients, which can be considered personal devices owned by voters, *e.g.*, typically web browsers fetching the voting client software from a voting server.

This allows for casting a remote vote in which the ballot is sent to the voting server to be persisted on a database that acts as a .

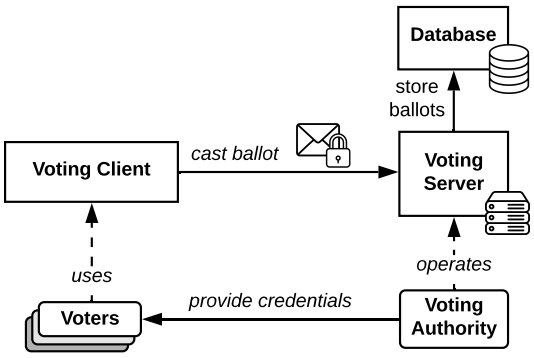


Figure 3.4: Centralized REV System's Architecture

For any properties to be evaluated, all trust assumptions need to be formulated beforehand. In essence, the centralized REV system requires a fully trusted VA that operates, stores, and manages cryptographic election keys to calculate the final tally. Further, the VA needs to operate a trusted server that persists all ballots cast and ensures the integrity and immutability of these ballots. While the server can be queried from the outside, the server's data is the single source of truth.

Assuming that a centralized REV system is built on a HE-based (Homomorphic Encryption) voting protocol, the homomorphic tallying process does not decrypt single ballots but adds the tally homomorphically and only decrypts the resulting tally. If the system is employing an MN (mixnet) voting protocol, the mixing ensures BP. However, without any additional mechanism, the system does not provide Receipt Freeness (RF) because the voter can produce a receipt of the encryption procedure and its randomization parameters. Subsequently, Coercion Resistance (CR) can also not be guaranteed since a coercer could force a voter to leak his/her credentials. Also, since the centralized VA generates the public election key, single ballots could be decrypted, thus breaking BP altogether. While this threat can be mitigated by using threshold cryptography and splitting up the key to different trustees, the voting server's storage would still present a Single-Point of Failure. For the decryption, the trustees would be required to interact with said voting server securely.

The ElGamal cryptosystem is the most popular choice for REV systems, used in both HE and MN-based schemes. It bases its security on the DDH (Decisional Diffie-Hellman) problem (*cf.*, Section 2.2.2). Therefore, the centralized approach does not provide Everlasting BP (EP) because an adversary can query the , break the DDH problem in the future, and restore the private election

key to decrypt all ballots to plaintext. Lastly, Unconditional Privacy (UP) is also not given since BP relies on a trusted VA, *i.e.*, if the VA is compromised or malicious, BP cannot be guaranteed. Thus, the *condition* for BP is trust in the VA.

All verifiability properties in any centralized approach rely heavily on trust assumptions, *i.e.*, these properties *rely* on trusted VAs to persist and safeguard data integrity. For instance, to achieve Individual Verifiability (IV), voters can query the central database and check whether their vote is included in the set of ballots. Achieving Universal Verifiability (UV) is significantly more complex in a HE or a MN-based scheme. For instance, the system can use a to prove the correct decryption of the final tally or the final mix step, which the electorate can verify. Any centralized approach does not provide eligibility verifiability as such, since it would require someone to verify that only eligible voters voted only once. Since the VA is trusted to provide credentials to all voters directly, the VA could create fake voters or cast votes with unused credentials. Without Eligibility Verifiability, the final tally is not trustworthy. The first property of End-to-End Verifiability (E2E-V) is Cast-as-Intended (CaI) verifiability, which requires a method for voters to verify whether the vote had been cast as it was intended, *i.e.*, whether the encryption was performed correctly by the voting device. The centralized approach does not provide any method to check whether the voting client's operations were correct and, hence, does not provide CaI verifiability at all. Nevertheless, the centralized approach can provide Recorded-as-Cast (RaC) verifiability, which can be checked by querying the central database and checking whether the ballot has been persisted correctly. Tallied-as-Recorded (TaR) (also referred to as Counted-as-Recorded, CaR) verifiability is given when the final proof of correct decryption verifies successfully.

3.2.2 Decentralized REV System

A major drawback of any centralized architecture is the dependency on trusting a single VA, *i.e.*, , the privacy and verifiability properties only hold under the assumption of a single trusted VA. Therefore, an approach for a fully decentralized and end-to-end verifiable architecture based on a public permissioned DL is provided here. The generic decentralized architecture for a REV system (*cf.*, Figure 3.4) leads to the observation that the significant difference with a centralized architecture is the decentralized deployment of the , assumed to be operated as a public permissioned DL. On the voter's side, the process is similar to the centralized variant, but the voter interacts *directly* with the DL by running their own DL node or by communicating with a DL node operated by a TTP. Also, all VAs should operate DL nodes. These nodes form the public permissioned DL network, which the public can query, but only trusted VAs can sign and append new entries to.

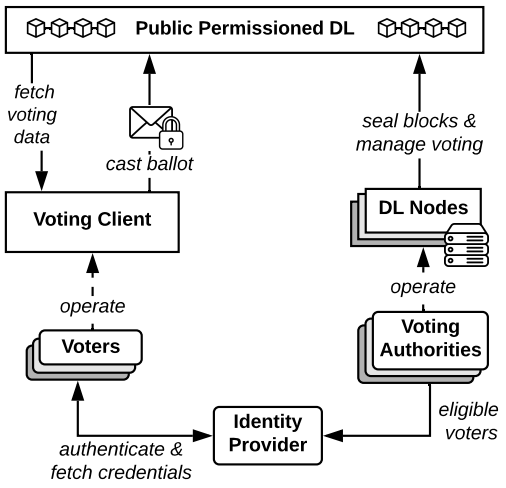


Figure 3.5: A Fully Decentralized and Generic High-level REV System's Architecture

The architecture (*cf.*, Figure 3.5) is agnostic toward any voting protocol selected. However, the voting protocol needs to include an approach that allows for the distribution of trust. By deploying a DKG scheme [109], the private election key is not held by a single authority. Finally, the authorities need to submit their decrypted share of the final tally to decrypt the result collaboratively [179]. In that sense, the public permissioned DL serves as a transparent, tamper-proof platform, allowing the voting protocol to execute publicly shared information, which the electorate and auditors verify. Given a voting protocol based on a public key cryptosystem, voters encrypt their ballots with the public election key and send the encryption to the DL network directly, where anyone can verify the inclusion of the respective ballots [179].

The major benefit of using a decentralized architecture is the use of a suitable permissioned consensus algorithm to decentralize trust in the . Many REV systems assume a centralized trusted that provides a consistent view of the election's evidence trail. The implementation of cryptographic schemes into practical systems deployed on centralized does not hold under real-world conditions and breaks verifiability claims (*e.g.*, integrity and availability guarantees). Thus, by decentralizing the , the risk of a single entity breaking the integrity and consistency of the is mitigated completely. The selection of an appropriate consensus algorithm is fundamental to the network's security characteristics [327]. For permissioned networks, where a set of initially trusted authorities is available (*e.g.*, VAs, Identity Providers, and other TTPs), the Proof-of-Authority (PoA) consensus algorithm is suitable. [175, 179] proposes such a PoA-based , allowing for the decentralization of trust. With a PoA consensus mechanism, the maximum number of tolerated malicious nodes is $\frac{N}{2}$ and $\frac{N}{3}$ for PBFT (Practical Byzantine Fault Tolerance), which is suitable for trusted VAs

in operation. The selection of PoA is aligned to those REV systems' general requirements (*cf.*, Section 3.1.2) since a set of trusted authorities is tasked with operating the voting or election. Thus, the authorities allowed to propose and sign blocks (also called *minting*) are defined *a priori*, enabling a bootstrapping stage where all authorities are already defined.

In summary, decentralizing the REV architecture allows for the distribution of trust on a public permissioned DL. The key benefit of a decentralized deployment is that trust assumptions are congruent with *(i)* trust assumptions of the voting protocol, *i.e.*, threshold cryptography allows for distributing trust among VAs, which generates a public election key, as well as decrypting the final tally collaboratively. Secondly, benefit *(ii)* is that the trust is decentralized on a DL-consensus layer, since the PoA consensus only accepts blocks from pre-defined authorities. Thus, the decentralized architecture maps trust assumptions directly to a deployed system. For that reason, the following section outlines detailed requirements to be addressed within the scope of a decentralized voting system.

3.3 Requirements of BC-based E-Voting

Requirements need to be defined for the design of any information system, especially if it involves complex cryptographic schemes, *e.g.*, a BC-based voting systems (*cf.*, Figure 3.6). Many of the following criteria are relevant for BC-based voting systems and crucial for EV systems in general, as outlined in [124, 125]. Transparency and public scrutiny are necessary to establish the public's trust in the voting system. For instance, allowing independent experts to examine the system and report vulnerabilities [224] critically. Therefore, essential security and transparency measures are highlighted as well.

3.3.1 Trust and Reputation in Voting Systems

Trust in the voting system is key to the stakeholders involved [178]. While "trust" is interpreted as a firm belief in the reliability, truth, or ability of someone or something to fulfill an agreement [107], reputation also plays an important role in this context, defined as a positively evaluated history of the interaction between agents. While a formal evaluation of reputation includes a protocol where interacting agents (*i.e.*, systems) rate their interaction, an informal evaluation relies on beliefs or opinions that are generally held about someone or something. For example, considering the reputation of external reviewers based on a history of previously and positively considered code reviews and reports.

Nonetheless, such informal notions provide ample room for interpretations, since *(a)* the criteria ranges from subjective to objective measures, *(b)* the value of truth is undisputed, but the truth itself

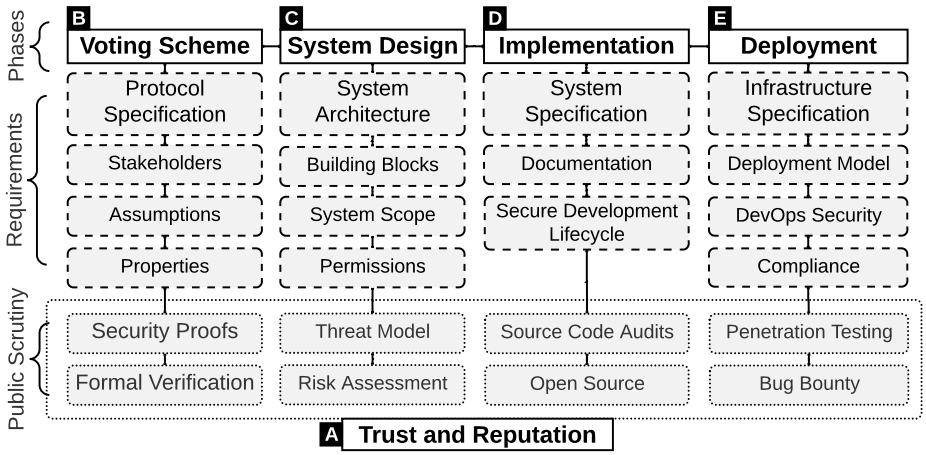


Figure 3.6: Overview of Major REV System Development Phases, Requirements, and Public Scrutiny Measures

is not always based on logical deductions in computer science, and *(c)* once applied in the context of voting systems, whether they are traditional paper-based systems or electronic, they have a close relationship to social and political sciences. The latter is especially relevant for EV systems, since their exposure in democracies stimulates reactions ranging from being democracy-undermining to being reliable, accessible, and correct.

Thus, when discussing voting systems, it is essential to address the following: *(1)* a pure technology-driven view is taken, described, and argued for, following a scientific methodology and a binary, logic-based set of assumptions, *(2)* the underlying technical mechanisms, algorithms, and processes must follow the current debate and understanding of cryptography-based approaches, and *(3)* any politically biased trust discussion is omitted, as the authorities, governments, or stakeholders involved play, and will always play, a major part in voting processes.

This means that an explicit deployment of reputation measures is not taken into account within this paper, since a risk analysis and risk mitigation fully address security risks in general, privacy risks in particular, and manipulation risks specifically. Therefore, citizens' judgments of trust in authorities, governments, and stakeholders in terms of voters and politicians are subject to other experts' work [202]. Furthermore, since [149] states that "Political trust is closely related to various forms of electoral behavior", *(a)* voters may be motivated to vote when they have trust, *(b)* they may be less motivated to cast a vote due to being distrustful, and *(c)* voting, in general, may boost or diminish the interest in elections.

In contrast, this work details discussions and overviews of technical specifics with the understanding that technological advances, such as the use or non-use of DL-based voting systems, can be quali-

fied in the context of trust assumptions made about the entire voting process. And these assumptions vary from country to country, from legislation to legislation, and from democracy to democracy, thus, determining not *the* solution, but a set of components that, through dedicated and carefully designed interplay, can form a secure, privacy-protected, and operational voting system with a fully electronic and DL-based approach driven by these specific defined trust assumptions.

3.3.2 Voting Scheme

A secure voting scheme is at the core of every voting system. The critical requirement for the voting scheme is the protocol specification, which contains models of the electoral system, the electorate, cryptographic building blocks, and parameters [124]. The formal specification also requires a definition of stakeholders and a trust model, *i.e.*, which stakeholder is entrusted with specific roles and responsibilities. The protocol specification also needs to mention cryptographic hardness assumptions explicitly. For instance, the ElGamal cryptosystem is often used in EV systems, and it assumes the Decisional Diffie-Hellman (DDH) problem to be computationally hard to solve [94].

In addition, the voting scheme's properties must be determined and evaluated. For instance, the precise definition of privacy and verifiability properties plays a fundamental role in assessing EV systems, which was performed in the following section. While formal security proofs evaluate these properties, additional methods to verify cryptographic protocols can also be applied (*e.g.*, Applied pi calculus [259]) to increase confidence in the voting scheme.

3.3.3 Voting System Design

Based on formal definitions, the system design develops the REV system architecture. A major building block is the voting software used by the voter and Voting Authority (VA) to encrypt and submit votes, query the , and verify proofs, as well as to collect, persist, and tally votes. Additionally, the identity management of voters is crucial. Depending on the electoral system, this role is either externalized (*e.g.*, via Identity Providers, IdP) or can be performed by the same stakeholder that takes on the part of the VA.

The system's scope maps the protocol specification to the system design and the electoral system [303]. *I.e.,* since the voting scheme and its protocol specification are generic, the system context maps generic assumptions and roles to specific instances embedded within the system's design. For example, within the protocol specification, a registrar is assumed to handle identity management. Thus, the system context can assign the role of the registrar to the IdP run by a state-owned company, which defines respective trust assumptions for the system's design.

To achieve a trustworthy voting system, its documentation needs to be published (asserting full transparency), and it requires an in-depth security assessment. Thus, a threat model must be established, and threat events must be classified and ranked within a risk assessment. Only such a risk assessment will yield mitigation strategies and further strengthens the public's understanding of the trust assumptions required for stakeholders and the voting system operator.

3.3.4 Voting System Implementation

In 2019, the Swiss Post published the source code for their sVote voting system. An international team of cryptography experts found failures in both the individual and universal verifiability based on flawed implementations of cryptographic components [134]. As a result, no version of the flawed system was used in Switzerland during May and October 2019. Since then, Swiss system providers have reiterated and published lessons learned about public scrutiny and requirements during the implementation stage [124, 224].

A key lesson is that the system's specification contains pseudo-code algorithms that map the protocol specification to the implemented code by following naming conventions (*e.g.*, method names, parameter names, and domain checks) [124, 302]. Furthermore, software engineering's best practices are applied with regards to testing (*e.g.*, Unit Testing, Property-based tests, and independent implementation test data) [302]. Following these best practices, thorough and consistent documentation of the source code is critical for a REV system for the modern age [224].

The correct implementation of software is of utmost importance and requires attention to security details at every stage. Also, manual source code audits must be an integral part of the Secure Development Lifecycle [150]. Additionally, transparency is not a "one-off measure", but a continuous process that needs to be used by the system provider [224]. For instance, publishing source code is an excellent way to get public scrutiny. However, to establish trust with the public and the expert community, publishing the code is not sufficient. At a minimum, critical parts of the code must be published under an open-source license (*e.g.*, cryptographic library) [224].

3.3.5 PBB Architecture and Deployment

The deployment of a REV system also requires precise specifications for the infrastructure, addressing relevant details for practical deployment. In the spirit of transparency and public scrutiny, infrastructure repositories need to provide a detailed demo setup that allows cybersecurity experts to deploy a local penetration testing setup. Also, the deployment model for different components of the voting

system needs to be documented and specified with regards to how the frontend is served to voters, how identities are managed, and how authentication methods are deployed.

The is a key component for every REV system since it keeps the audit trail of the election and provides a tamper-proof view of the system's state. Major options for deployment models of s are depicted in *cf.*, Figure 3.7 [177]. Centralized deployments are characterized by a single data source managed by a single centralized organization. In order to mitigate the risk of centralization, distributed deployments allow for a single organization to achieve higher availability and distribution of storage. However, this distribution requires consensus to achieve consistency of the data stored [303]. Furthermore, decentralized deployments allow *N* organizations (*e.g.*, cantons, municipalities, or states) to run *N* servers with an underlying consensus. For instance, a public permissioned DL with Practical Byzantine Fault-Tolerance (PBFT) is classified as a . Lastly, a permissionless PBB deployment is classified as a Public BC. For instance, deploying a fully Smart Contract (SC)-based voting system on a public BC (*e.g.*, Ethereum Mainnet) is considered a permissionless and decentralized deployment. From centralized to decentralized deployments, these methods are not restricted to only the architecture. They also include overall system design choices, for instance, the Identity Management System as well as various frontend and backend services (*e.g.*, whether the frontend is served from a centralized entity).

DevOps is described as the *conceptual and operational merging of development and operation needs, teams, and technologies* [56]. Furthermore, the DevOps security (*i.e.*, DevSecOps [227]) needs to be considered by including security considerations early on in the project's DevOps cycle. Thus, the need for security in DevOps is met by DevSecOps[227], which extends DevOps by involving security experts from the start [223].

Finally, to build trust and reputation in a voting system, "Bug Bounties" and "penetration testing" are also needed. On the one hand, depending on the scope and deployment of the target software component, a Bug Bounty program can leverage the global community of cybersecurity researchers to search for vulnerabilities and weaknesses. Bug Bounties are helpful for clearly defined scopes and system boundaries. On the other hand, penetration testing allows for a more openly defined scope and enables lateral movements in the network (*e.g.*, privilege escalation across multiple domains, servers, or networks).

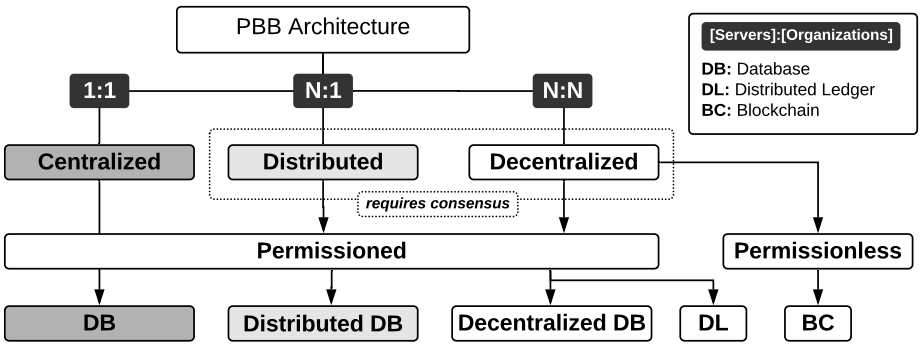


Figure 3.7: Deployment Models for PBB Architectures [177], the black indices describe physical or virtual number of [Servers] and [Organizations] *i.e.*, trusted authorities participating in the consensus, *e.g.*, Distributed means N servers controlled by a single organization.

3.4 Overview of REV Approaches

Besides the specific requirements outlined previously, the most crucial voting scheme approaches used in most REV systems will now be highlighted. Over the years, many REV approaches have been developed, evaluated critically, and refined. Thus, it is essential to describe these approaches, outlining their core ideas and adding necessary elements on selected cryptographic elements that may not have been covered in previous sections, providing the required background to follow the principle approaches outlined hereafter. Inspired by [123] and [28], the following figures' organization serves as an essential reference and model to introduce REV schemes and their specific challenges. The following approaches highlight the inherent problems with creating an REV system in practice. Privacy, verifiability, and trusted parties are given access to overview major features, characteristics, and concerns that are still partially present. While the is an essential part of any of these approaches (and REV systems in general), the following focus is on the approaches, regardless of the practical deployment methods used (*e.g.*, be it a centralized server, a BC, or a DL).

3.4.1 Encrypt and Sign Approach

A basic approach towards an EV scheme combines an encryption scheme with a digital signature scheme, as outlined in Figure 3.8. The encryption scheme provides ballot privacy, while the signature scheme ensures the data integrity of the sender's transmitted message and authentication. During ongoing voting, the secrecy of the voters' ballots is ensured. However, since the voting service has the private election key (*i.e.*, the counterpart of the public election key used to encrypt the ballots), the digital signatures directly link the encrypted ballots to the voters [123].

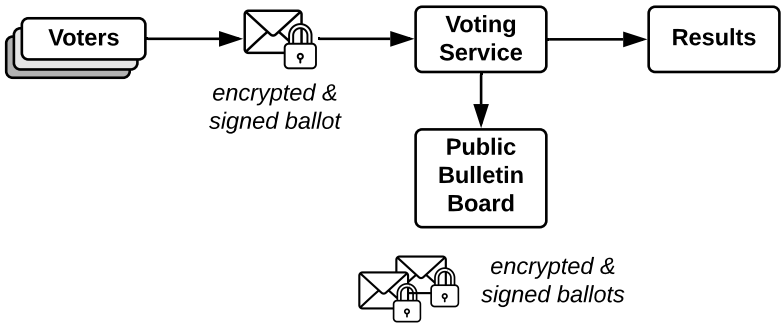


Figure 3.8: Basic Remote E-Voting Architecture based on a Simple Encrypt and Sign

Thus, the voting service acts as a single trusted entity and can break ballot secrecy individually (*e.g.*, decrypt individual ballots without anyone realizing). Also, a breach of the voting service's security could potentially leak the private key for the election and allow for complete decryption of the , breaking ballot secrecy in full. By publishing a list of eligible public voter credentials, anyone can verify the signatures on the ballots. However, the major issue is that the final results can not be verified without any additional mechanisms because the voting service would break ballot secrecy by decrypting ballots and publishing their plaintexts. While individual voters can verify whether their ballot was stored correctly on the , they cannot verify the final results and fully trust the voting service. In summary, the encrypt & sign approach is feasible for scenarios where voters choose to trust the voting service fully.

3.4.2 The Two Agencies Approach

Another popular voting scheme was first proposed in 1992 by [104], often referred to as the *two agencies model* [247], FOO92, or just FOO [28]. It is one of the first and most influential REV schemes designed for practical use. A simplified version of this voting scheme is shown in Figure3.9. The Validation Service authenticates the voter (checks eligibility) and then provides an anonymous token to the voter. The voter sends the encrypted ballot with the anonymous token to the Voting Service, which only appends the ballots if the Validation Service issued the anonymous token. The introduction of the validation service as a new TTP removes the link between the encrypted ballots and the voters' identities.

Voting schemes based on the two agencies model usually use the cryptographic primitive of Blind Signatures, which allows a signer to cryptographically sign a message without viewing its content in plaintext. Blind Signatures were first introduced in [49] as a means to establish untraceable and

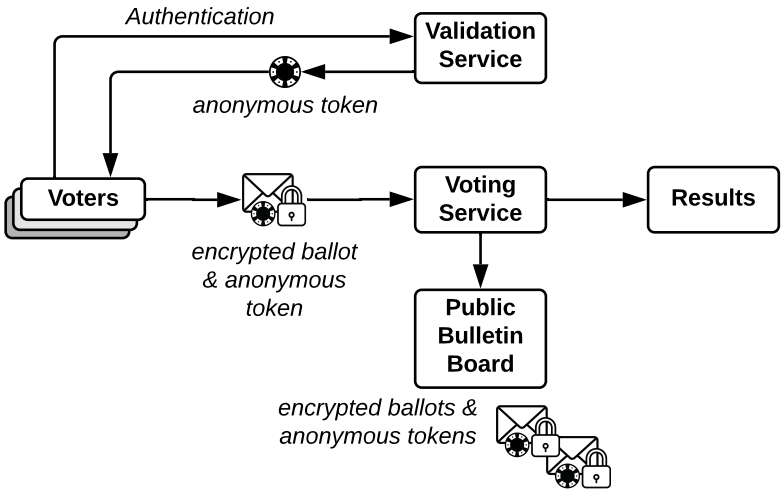


Figure 3.9: Two Agencies Model with Anonymous Tokens (*e.g.*, Blind Signatures)

anonymous electronic payments. In the context of Figure 3.9, the Validation Service can digitally sign the ballot without viewing its content (*i.e.*, signing the blinded ballot). The voter receives the signed ballot, removes the blinding factor, and then casts the unblinded but signed ballot to the Voting Service, which validates whether the signature originates from the Validation Service. Even if the Validation Service colludes with the Voting Service, the token can not be traced back to the voter since it was signed in its blinded form. Ballot Secrecy is ensured in the final tally along the same line of argument because the link between the vote and the token is removed. However, collusion between the two trusted agencies would make it rather easy to identify voters and break ballot privacy by correlating the information. Another major threat is *ballot stuffing* because the Validation Service could create valid tokens for non-eligible voters, which the Voting Service would accept. With this scheme, the voting service is still able to decrypt individual ballots since it owns the private election key.

3.4.3 Homomorphic Encryption Approach

Another major approach to achieving a private and verifiable REV is deplying HE. An exemplary homomorphic tallying system is depicted in Figure 3.10. The ballot must be encrypted according to the format defined by the voting service. For instance, in this case, three ciphertexts of binary values, either 0 or 1, are encrypted three times, once for each voting option *A*, *B*, and *C*. Additionally, the ciphertexts may be cryptographically signed (for eligibility verification), and usually, the cryptographic proof is generated (*e.g.*, a ( ). More on that in Section 2.9). Because not a single ciphertext is decrypted during tallying (only the result), the proofs are crucial to ensure the ballot's validity and to

mitigate the risk that malicious voters could encrypt invalid values. Therefore, during tallying, the underlying cryptosystem's homomorphic properties are used to add the ciphertexts together and then only decrypt the final result. The major drawback of HE-based systems is that the number of possible voting options is limited (*e.g.*, write-ins can not be supported). A precise representation of the ballot is required.

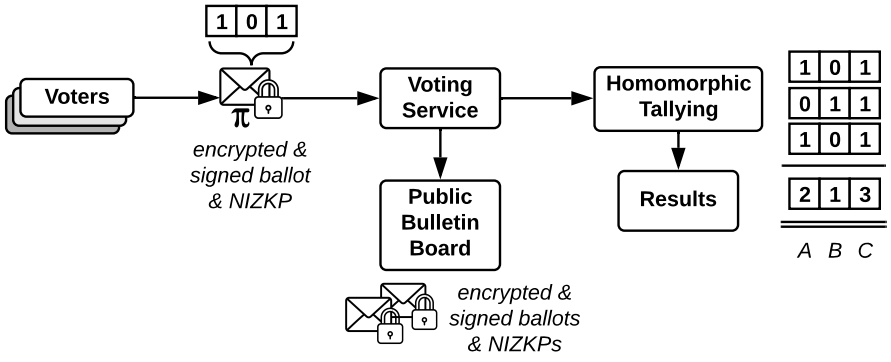


Figure 3.10: Homomorphic Encryption-based REV System (*e.g.*, ElGamal)

Despite these limitations, an interesting advantage of HE-based systems is the straightforward distribution of decryption keys by using threshold cryptography. In such a case, each trustee holds shares of the private key that is required to decrypt the final tally. In 1997, the seminal work [65] proposed the use of threshold cryptography in a REV system considering a multi-authority setting that requires each key share holder to submit partial decryptions. Then, the final decryption can only be performed if the predefined threshold of authorities collaborates.

An example voting system as depicted in Figure 3.10 usually requires trustees to perform a DKG algorithm, which results in a public election key $_{e}$, while each trustee holds a secret key share $_{e}$. Then, voters form a valid ballot and encrypt their ballot by using $_{e}$. Further, voters generate a proof proving that the ballot is well-formed and valid before sending both to the voting service. Upon receiving the ballots, the voting service authenticates voters and validates the proofs, ultimately storing them in the . After the voting period ends, the homomorphic tallying is performed among all trustees over the set of all stored and valid ballots. For that, the trustees use their secret key shares to tally the ballots together homomorphically (without decrypting individual ballots), only decrypting the final tally and publishing it to the public.

3.4.4 Mix Network-based Approach

Besides HE, mix networks (MN) are the primary building blocks for deployed REV systems used in real-world elections in Switzerland, Australia, Estonia, and Norway [135]. MNs are versatile cryptographic protocols that can be tailored to other privacy-sensitive use-cases, such as anonymous communication protocols for messaging or routing.

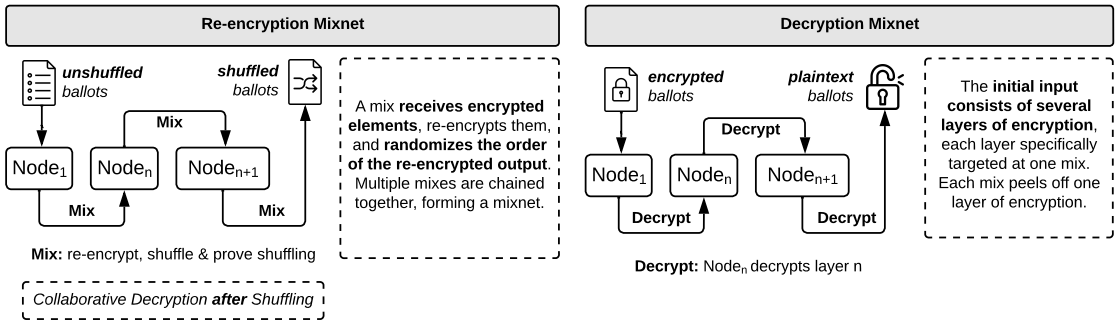


Figure 3.11: Mix Network-based REV System

In principle, MN-based approaches are similar to paper-ballot voting schemes where the paper ballots are shuffled after they are separated from any voter-identifying information. As seen in Figure3.11, once the ballot is no longer traceable through the MN, correlating the ballot before and after the MN is impossible. For the final tally, each ballot is decrypted individually in contrast to HE, where only the ciphertext of the final tally is decrypted. Further, MN-based schemes can distribute trust by leveraging threshold cryptography (similarly to HE-based schemes *cf.*, Subsection 3.4.3), for instance, by using multiple trustees to perform the MN re-encryption and final decryption ceremony.

E-Voting: A topic that cryptographers like to publish papers about but don't like to see deployed in reality because it's perceived as unacceptably risky.

Jean-Philippe Aumasson, 2021 [11]

4 Literature Review

The analysis of state-of-the-art of voting systems and schemes lays the essential foundation for the design of voting scheme improvements and novel voting system designs. This thesis mainly divided these systems into Remote Electronic Voting (REV) systems and Remote Postal Voting (RPV) systems. REV systems focus on remote voting from any personal device using a dedicated voting application or web application that is submitted over the internet. On the other hand, RPV systems are paper-based. Further, this thesis focuses on decentralized schemes (*e.g.*, based on DL or BC), where advanced cryptographic mechanisms are leveraged to distribute trust. Although most cryptographic voting systems have conceptual overlaps with each other, their use case and assumptions differ and need to be looked at from dedicated points of views.

A keyword analysis (*cf.*, Fig. 4.1) performed on research published in this area indicates a steady increase in academic activity stemming from the topics covered by this thesis. The keywords were selected based on the most important architectures (*i.e.*, BC, DL), deployments (decentralized and distributed), and in combination with its application (*i.e.*, voting). It is important to mention that the decrease in 2022 may falsely indicate a decrease in interest because the queries were aggregated in September 2022. In contrast, a lot of the listings occur later in the year and in early 2023.

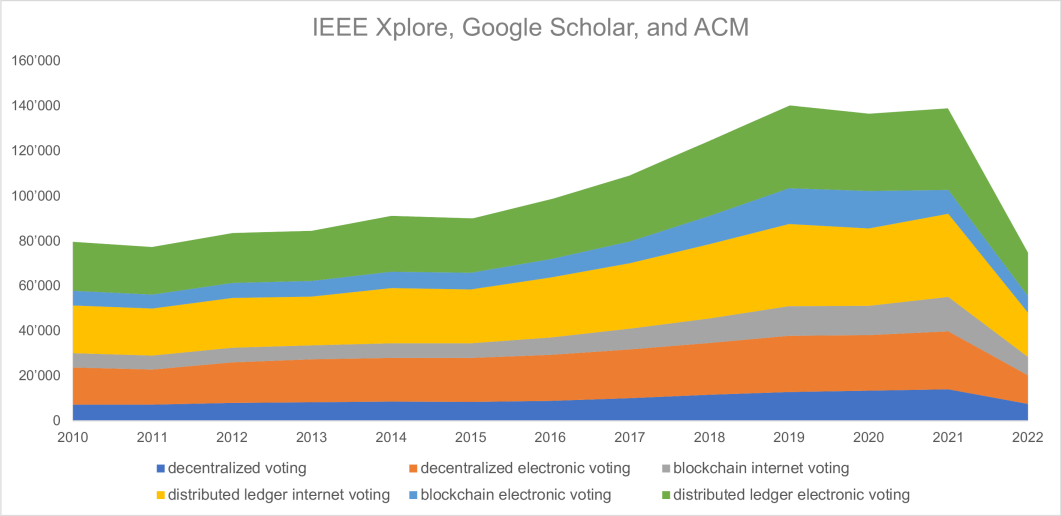


Figure 4.1: Number of Search Results by Keywords per Year (IEEE Xplore [152], Google Scholar [118], and ACM [9])

4.1 Review of Remote Electronic Voting Systems

Since the inception of public-key cryptography and modern cryptograhic algorithms in the eighties, cryptographers have been tinkering with protocols to implement private and verifiable elections.

In Table 4.1, the most relevant related works are classified following a bottom-up approach, where the key underlying academic protocols are first summarized, following their evolution to full-fledged country-wide election systems. It is important to note that the following review focuses on *remote* EV systems. Although the first category of *Academic Protocols* can be adapted for use in on-site settings, this review focuses on protocols that are feasible for use in REV systems. For a complete overview, please refer to [158].

In cryptography, defining the threat model or trust assumptions is crucial. Therefore, it is essential to define trust assumptions for the comparison performed in this literature review (*cf.*, Table 4.1). For instance, if all authorities collude in a multi-authority protocol, they could decrypt all ballots and see the votes in plain text, and BS would be violated. Hence, it is important to define the underlying trust assumptions as follows:

- At least one authority in a multi-authority protocol is trusted.
- The voting device is not trusted.
- The PBB is trusted to store and return values honestly.
- The voter and the randomizer (if there is one) do not collude.

The first category are *Academic Protocols*, *i.e.*, protocols that were proposed mainly by cryptographic researchers and published as research papers, and are not accompanied by any detailed system specification or implementations, while the second category are *Academic Implementations*, which are voting protocols that also showcase practical feasibility with working implementations and Proof-of-Concepts (PoC). Lastly, *Enterprise Implementations* are reviewed, where a selected few are presented, since oftentimes the details are not available publicly for review, or the privacy and verifiability properties are not verifiable. Finally, *Country-Specific* implementations are reviewed. Many countries have piloted REV systems, but most of them have halted these efforts due to security concerns.

4.1.1 Multi-Authority protocol CGS97

Cramer, Gennaro and Schoenmakers (CGS97) published a seminal work in 1997. Figure 4.2 depicts an overview of different implementations that are based on the CGS97 protocol. Certainly, the systems depicted in 4.2 do have other major influences on their design and architecture choices, however, the major decisions on the multi-authority setting and distribution of trust have their origins in the scheme proposed in CGS97.

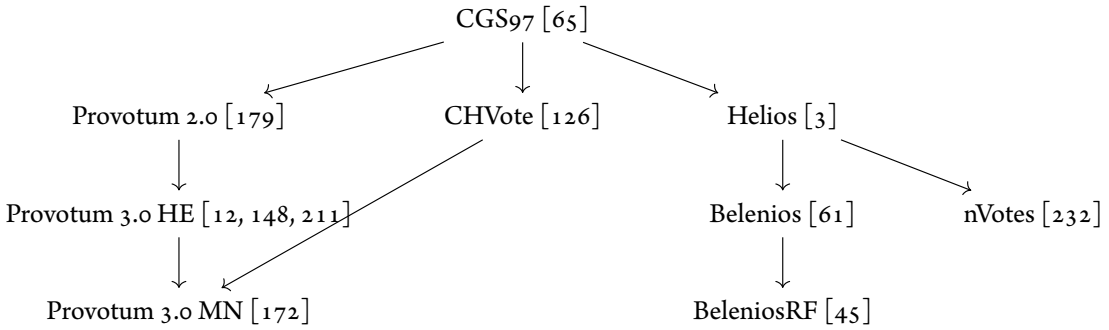


Figure 4.2: CGS97 Domain Protocol System and its Influence on the Implementations that followed

In CGS97 [65], an efficient voting scheme tailored towards multi-authority settings was proposed, reducing the trust assumptions in invdidual authorities [158]. The scheme is quite influential and led to various adaptations and extensions towards RF. In CGS97, voters first encrypt their voting choice with the election public key, and then generate an accompanying ballot validity , which are both stored on the PBB. The proof can be turned into a with the Fiat-Shamir heuristic [100]. In [65], public verifiability and communication is achieved through the PBB. Further, the scheme claims to provide UV and BP, but no RF (*cf.*, Table 4.1).

	Ballot Privacy	Receipt Freeness	Coercion-Resistance	Everlasting Privacy	Unconditional Privacy	Software Independence	Individual Verifiability	Universal Verifiability	Eligibility Verifiability	Cast-as-Intended	Recorded-as-Cast	Tallied-as-Recorded	Transparency	Decentralized PBB	Voting Protocol
Academic Protocols															
CGS97 [65]	●	○	○	○	○	●	●	●	○	○	●	●	NA	NA	HE
JCJ05 [160]	●	●	●[1]	○	○	○	●	●	●	○	●	●	NA	NA	RMN
HRZ08 [138]	●	○	○	○	○	●	●	●	◐[2]	○	●	●	NA	NA	HE
Academic Implementations															
Helios [3]	◐[1]	○	○	○	○	●	●	●	○	●[2]	●	●	●	○	HE
BeleniosRF [46]	●	●	○	○	○	●	●	●	●[4]	○	●	●	●	○	HE
Civitas [54]	●	●	●	○	○	○	●	●	●[4]	○	●	●	●	○	RMN
Selene [260]	●	●	○	○	○	●	●	●	○	●[5]	●	●	◐[6]	○	RMN
Verify-Your-Vote [47]	●	●	●	○	○	●	●	●	●[4]	○	●	●	◐[6]	●	HE
OVN McCorry [215]	●	○	○	○	○	●	●	●	◐[7]	○	●	●	●	●	HE
Chaintegrity [329]	●	○	○	○	○	○	●	●	●	○	●	●	○	●	HE
BBB-Voting [309]	●	○	○	○	○	○	●	●	●	○	●	●	●	●	HE
Provotum 2.0 [179]	●	○	○	○	○	●	●	●	●	○	●	●	●	●	HE
Provotum 3.0 RF [148]	●	●	○	○	○	●	●	●	●	○	●	●	●	●	HE
Provotum 3.0 CAI [211]	●	●	○	○	○	●	●	●	●	●	●	●	●	●	HE
Provotum 3.0 MN[13] [172]	●	●	○	○	○	●	●	●	●	○	●	●	●	●	RMN
Æternum [173, 178]	●	◐[3]	○	●	●	●	●	●	●	○	●	●	●	●	CS
Enterprise Implementations															
nVotes [232]	●	○	○	○	○	●	●	●	○	●[2]	●	●	○[8]	○	RMN
Voatz [286]	○	○	○	○	○	○	○	○	○	○	○	○	○[8]	◐[12]	n/a
Vocdoni [315]	●	●	○	○	○	○	●	●	●	●	●	●	●	●	BS/ZKP
TIVI [279]	●	●	○	○	○	○	●	●	●	●	●	●	◐	●	RMN
Country-specific Implementations															
CHVote [126]	●	○	○	○	○	○	●	●	●[4]	●[9]	●	●	●	○	RMN
Estonia [144]	●	○	○	○	○	○	●	●	○	●	●	○	◐[10]	○	RMN
Norway [112]	●	○	○	○	○	○	●	○	○	●	○	◐[11]	●	○	HE

[1] Vote copying possible
[2] Cast or challenge
[3] Extension available in [203]
[4] Uses trusted registrar
[5] Referred to as ProvotuMn
[6] Used e2e tracking number
[7] No source code available
[8] Needs trusted contract creator
[9] No source code, no documentation available
[10] Mailed verification code
[11] Partial source code available
[12] Needs trusted auditor
[12] BC deployed in private cloud

Table 4.1: Comparison of Related Work on REV Systems ● = Provides Property, ○ = Does Not Provide Property ◐ = Partially Provides Property.

Helios [3] is an open-source, web-based REV system that anyone can use to run their election [3]. The protocol is based heavily on CGS97. It uses DKG for a multi-authority setting and homomorphic encryption as outlined in CGS97, as [3] does not claim to add new concepts to the base protocol but rather implements a practical system that can be used in real-world elections [158]. However, it is noted that it should be used in low-coercion environments, like in small university or association elections [3]. From a privacy perspective, *Helios* does not fulfill RF, CR, and only partially fulfills BS due to the replay attack described in [62].

Evaluating its verifiability properties, *Helios* implemented a challenge-or-cast mechanism to fulfill the cast-as-intended property. Recorded-as-cast is fulfilled since the voter can observe his encrypted ballot on the PBB, and tallied-as-recorded can be shown by the distributed decryption of the sum of the votes (including right decryption proof from the authorities). Moreover, *Helios* does not fulfill eligibility verifiability. Voters must trust the voting authority that only eligible voters are accepted [196]. *Helios* was already used in small but legally binding elections within academic contexts, like a presidential election at the Universite Catholique de Louvain in 2009 [162].

Belenios [61] is a web-based REV systems that aims to fulfill BS, E2E-V, and eligibility verifiability [61]. The protocol builds heavily on the *Helios* protocol. First, *Belenios* adds a trusted registrar, which is responsible for distributing a public-private key pair to all the eligible voters (using an authenticated or an offline channel). The voter then uses the private key to sign the ballot. All public keys are published on the such that everyone can verify that only eligible voters have cast ballots [60]. Under the assumption of a trusted registrar, *Belenios* additionally fulfills eligibility verifiability. Second, since the ballot is signed by the voter, a replay attack using the same ballot as another voter is no longer possible [61]. Third, *Belenios* does not implement a challenge-or-cast functionality like *Helios*, since the authors claim that the challenge functionality was not used often in practice and voters that don't do the challenge are easy targets for attacks [61].

BeleniosRF [45] builds on top of Belenios. As the name already indicates, *BeleniosRF* adds the RF property to the system. In [45], a re-encryption randomizer for ballots is introduced. Ballots are sent to this randomizer before casting [45]. As long as the voter and randomizer do not collude, the voter can not reproduce his encrypted ballot anymore and, therefore, does not have a receipt of his voting decision.

4.1.2 Decentralized and Distributed

A crucial element investigated in this thesis is the deployment type of the . The following systems are academic implementations that implement decentralized or distributed .

Provotum 3.0 RF [148] (also referred to as *ProvotumRF*) is built on the initial design of Provotum 2.0. The protocol is extended with a re-encryption randomizer component that re-encrypts the voter's encrypted ballots. Similar to *BeleniosRF*, the voter can not reproduce her ballot anymore because the randomization for the ElGamal encryption is not available on the voting device. Under the assumption that the randomizer and the voter do not collude, *ProvotumRF* additionally fulfills RF. Additional extensions to Provotum 3.0 include a decentralized and self-sovereign identity management system [12], as well as Cast-As-Intended verifiability[211], which is based on Benaloh's Challenge-or-Cast mechanism [19].

Provotum 3.0 MN [91, 172] is an evolution of *ProvotumRF* that changes the core voting protocol from using HE to a RMN, which is executed in a fully decentralized fashion using a DL. In the post-voting phase, the encrypted ballots are shuffled (mixed) off-chain to remove the connection between the voter and her ballot. This involves multiple rounds of shuffling. The sealers generate (which are later validated by the) to prove that shuffling has been performed correctly, as this process is executed off-chain. Afterward, the sealers perform CD on the encrypted ballots. As no HE is used for the final tallying, the sealers publish their partially decrypted shares to the . Note that the plaintext votes are only publicly displayed after the result of the vote has been published [172].

Æternum [173, 178] presents a practical implementation of the protocol proposed in [203]. The novelty of the system is a fully decentralized voting system that achieves Unconditional Privacy, *i.e.*, there are no computational intractability assumptions related to BP. Computational intractability is only relied upon to ensure voters submit valid values to the during the voting phase. As a consequence of not encrypting the ballots, BP does not rely on computational intractability assumptions, thus achieving UP and EP. This comes at the cost of fairness, as it is possible to calculate partial tallies before the voting phase has closed. From a verifiability standpoint, Æternum achieves IV and UV, as ballots are stored on the and can be verified by anyone, but it lacks CAI verifiability, which is needed to provide E2E-V.

nVotes [232]: nVotes is a web-based REV system developed by the Spanish company Agora Voting SL [232]. Only minimal information about the protocol is publicly available. However, del Blanco et al. [72] had insights into the protocol and were able analyze some privacy and verifiability properties.

According to del Blanco et al. [72], the voting protocol is based heavily on *Helios* and starts with a DKG among multiple authorities. A public key for the election is then generated. Voters can then log in to the web platform using SMS authentication. After selecting their voting choice and generating the encrypted ballot, a challenge-or-cast mechanism is available to ensure cast-as-intended verifiability. Before the tallying phase, the votes get mixed with a Verificatum mix-net [322]. In the last step, partial decryption is employed, and the votes are tallied. According to [72], the protocol fulfills

E2E-V under the assumption that many voters use the challenge-or-cast functionality and that the authorities do not collude [72]. Congruent with Helios, RF and CR are not fulfilled. According to the website, more than two million votes have been cast in total, and they have processed more than 150,000 votes within a single election [232]. Clients range from public administrations (Barcelona Provincial Council, Madrid City Council) to education institutions (UNED) [72].

4.1.3 Coercion-Resistance of JCJ05

Coercion-Resistance is a privacy property first introduced in ***Juels, Catalano, and Jakobsson (JCJ05)*** [160], which also proposed a new protocol that satisfied the property. The protocol uses a multi-authority scheme with DKG, a RMN to disconnect ballots from the voters, and private credentials generated by a trusted registrar [160]. The seminal work spawned refinements and novel improvements (*cf.*, Fig. 4.3). As previously outlined in Section 3.1.1, [160] refined the notion of CR. JCJ05 highlighted that *(i)* forced abstention, *(ii)* forced credential hand-over, or *(iii)* forced randomized voting are not all covered by the privacy notion of RF, thus proposing CR to also prevent these three attacks. It is important to mention that [160] laid the foundation for other systems, *e.g.*, Civitas [54]. The assumptions of [160] include an untappable channel between registrars and voters, as well as an anonymous channel from the voters to the PBB.

The reason this protocol is CR lies in the fact that a voter can submit ballots with fake credentials. After the mixing (when ballots and voters can not be mapped anymore), these fake ballots are removed by the protocol and it is no longer possible to determine who cast the fake ballot [158]. With this functionality, the voter can use fake credentials during coercion and repeat the voting with a valid ballot when she is not coerced. Therefore, the voting protocol also fulfills receipt-freeness (the voter can not prove if she used fake or valid credentials) and ballot-secrecy (the decrypted ballot is not connected to a voter). Besides CR, the scheme also claims UV and RF. The protocol does not fulfill E2E-V since CaI is not proven within an environment of compromised voting devices.

Civitas [54]: Civitas is the first known implementation that fulfills the strong notion of CR [158]. The project implemented the JCJ05 protocol with some adaptions. [54] implement a registrar and different registration tellers to distribute trust in distributing the private credentials. The registration tellers only generate a share of the final voters' private credentials. The voter then requests all shares from the registration tellers and combines them with his full private credential. With this functionality, only the voter owns the full plaintext of the private credentials (as long as not all registration tellers collude) [158].

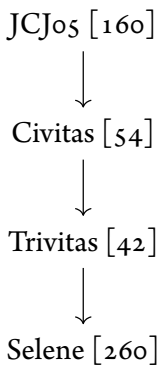


Figure 4.3: JCJ05 Protocol and Dependent Implementations

A key disadvantage of the implementation is the fact that the removal of invalid ballots with a Plaintext Equivalence Test (PET) requires quadratic computation time [231]. However, newer approaches to this problem have already reduced the computational time complexity from quadratic time to linear time [290, 321]. Civitas also fulfills UV since all operations on the can be verified by anyone. for correct shuffling and mixing, as well as for the PET for removing invalid votes, and for correct decryption can be checked on the [42].

Trivitas [42] aims to improve IV based on the implementation of Civitas. The authors criticize the complex mathematical operations required for a voter to verify that his ballot is included in the final published set of votes within popular implementations like Civitas or *Helios*. Moreover, these implementations normally require that a voter perform the verification on a trusted device.

Trivitas introduces the notion of a trial ballot. The trial ballot contains trial credentials that can be cast like a valid or fake ballot but will not be counted in the final tally. A flagged trial ballot can be used as an audit ballot in three different voting phases. Before the mixing, trial ballots are decrypted and posted on the such that the voter can ensure that the votes have been encrypted and recorded correctly on the . Next, the set of trial ballots also get mixed with the valid and fake ballots and posted on the . The voter can ensure that all his votes have been used as an input for the RMN. Finally, trial ballots are decrypted and posted on the after the mixing. The voter can ensure that the vote occurs on the election outcome. With this functionality, the voter can track the trial ballot along with the voting protocol and can ensure that it works properly [42]. The idea of a Trivitas' trial ballot is similar to a Challenge-or-Cast ballot. However, it does not get discharged after the ballot creation but is instead tracked during all steps of the voting protocol. Trivitas adds the fulfillment of CaI to the base protocol of JCJ05 because the trial ballot can be audited in the decryption step before the mixing, thus a voter can check that a trial ballot correctly recorded his choice on the .

Selene [260]: Selene is a voting protocol based on JCJ05, but uses a private tracking number for each voter. The intention is to make it easier for a voter to verify end-to-end. The main idea is that decrypted ballots are published on the along with a public tracking number. The voter can then easily check if her vote and the corresponding tracking number are included in the published list. The protocol implements the functionality using a private (α) and a public (β) part of the tracking number. The private part is transferred to the voter, and the public part is published on the . The voter can then combine these two parts and decrypt them using his private key to get the public tracking number. It is also possible for the voter to fake α such that her decryption results in another tracking number being published along with the decrypted ballots.

With this property, it is not possible for a voter to prove how she voted to someone. Hence, the protocol fulfills RF. It is not CR since a coercer could watch the voter casting his vote. However, it can reduce the coercion risk since the voter can present an incorrect tracking number to the coercer. The protocol fulfills E2E-V since to voter can create his tracking number from the true α and β and check if the vote was correct in the final list of tallied ballots. Eligibility verifiability was not considered in this protocol [260].

4.1.4 Self-Tallying Scheme HRZ08

Hao, Ryan, and Zielinski (HRZ08) [138] proposed a self-tallying, two-round, distributed voting protocol. The core idea is that no central authority is required for protocol execution, and the tallying is enforced by the protocol itself (thus the name *self-tallying*).

HRZ08 fulfills BS since nobody can decrypt a single vote. However, RF and CR are not given. The voter can prove how she voted by decrypting her vote. E2E-V is also not achieved since cast-as-intended can not be ensured if a voting device is compromised. Eligibility verifiability was not considered in this protocol. Moreover, the protocol assumes that every eligible voter that performed the first round also needs to perform the second round. Otherwise, it is not possible to calculate the final tally. Figure 4.4 illustrates the protocol for HRZ08, which will be elaborated on below.

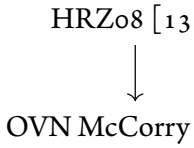


Figure 4.4: HRZ08 Domain Protocol and its Dependent Implementation

OVN McCorry [215]: OVN McCorry is a direct implementation of the HRZ08 domain protocol. The implementation uses a smart contract on the Ethereum blockchain [43] as the . A central authority generates a smart contract, defines public parameters, and adds all the eligible voters. The voters then perform the two-round HRZ08 protocol. The implementation fulfills the same properties as the domain protocol but also adds eligibility verifiability to some extent. If the central authority is trusted and only eligible voters are included in the smart contract, everyone can check that only eligible voters cast votes to the final tally. However, the PoC implementation also has some usability issues. The use of SCs on the Ethereum BC only allow for approximately 60 votes. Otherwise, the maximal transaction fee would be too high [215].

4.1.5 Commercial and Country-Wide Deployments

The systems contained in the following category are either commercial products or tailored to a country's specific requirements. Therefore, the following systems can not be clearly attributed to one of the academic protocols, although they may have similar designs and voting schemes.

Vocdoni [315] is a commercial REV system built on top of two distributed systems. While it uses the Ethereum BC to keep track of running elections, Vocdoni uses a privately operated PoA DL called Vochain, which is responsible for storing the encrypted votes. In terms of IdM, it supports both centralized and SSI-based [245] approaches. Vocdoni achieves BP by using BS or ZKPs. Fairness is achieved by revealing the private election keys after the last vote has been cast. The use of public-key cryptography implies that EP and UP are not fulfilled. The fact that the user has access to a so-called "nullifier", which identifies her vote, indicates that RF is not achieved. That a voter can only cast her vote once for a particular election suggests that CR is not achieved either. In terms of validation, Vocdoni provides E2E-V.

TIVI [279] is a commercial REV system that leverages a on a permissioned BC. While the exact steps of the voting protocol are not publicly disclosed, the properties and some of the cryptographic primitives used are publicly known. In order to achieve BP, TIVI applies asymmetric encryption to ballots. Digital signatures are applied to the ballot to guarantee the integrity of the vote. When casting the ballot, it is forwarded to a MN in order to disguise the relationship between a voter and her ballot. For tallying, the electoral board uses CD to decrypt the ballots, implying that DKG must have been used beforehand. Fairness is fulfilled by the fact that the final tally only is visible after the electoral board has performed the vote counting. TIVI provides CR, as it allows one to re-cast the ballot, considering only the last ballot from each voter for the final tally. UV and E2E-V are fulfilled, as tools are provided for the verification of the ballot on the BC [279].

Voatz [286] is the first REV system used in US federal elections. It is a smartphone app that should provide a simple user interface to cast votes. Moreover, a BC is used as a . Since Voatz does not provide much information about their protocol, this analysis is based on [286]. According to [286], details on the server infrastructure were not provided and therefore assumptions had to be made. Furthermore, the smartphone app had to be reverse engineered. [286] found that the app had several security vulnerabilities and serious privacy issues. In addition, the server had complete control over the protocol and did not include any proofs. Hence, the VA could potentially include fake votes or delete votes without detection. Thus, the VA is fully trusted with any privacy or verifiability properties. In addition, [286] found vulnerabilities in the non-standard encryption scheme between the communication of the app and the server. The encryption scheme has the property where the encrypted ballot has a similar size to the ballot in plain text. Therefore, someone could guess the plain text merely by seeing the encrypted ballot, *i.e.*, side-channel attacks are possible, revealing the voter's ballot. As a consequence, fairness is not guaranteed. Regarding verifiability, no indication of E2E-V was found. The fulfillment of RF is not clear, as there seems to be a receipt for voters, but its use is not clearly stated. CR is not provided. EP and UP are not achieved, as computational intractability is assumed for the AES encryption, and additionally, BP is not achieved in as outlined in [286].

Estonia [143]: The Estonian REV system used in public elections between 2005 and 2015 did not provide strong verifiability. [143] proposed improvements in verifiability to the base protocol. The new protocol provides a PBB and is based on a MN. Voters can encrypt their ballots using the election public key, sign them using their private key, and send them to the . To provide cast-as-intended and recorded-as-cast, the voter can request his ballot from the system, locally generate the ballot for all options again (with the same randomness), and compare these ballots with each other. When the encrypted ballot is the same, she can ensure cast-as-intended and recorded-as-cast. However, receipt-freeness is therefore clearly violated. After the voting phase, the ballots get mixed and decrypted. Decrypted votes, proofs of correct decryption and mixing, and the eligible voter's public keys are published on the . This ensures tallied-as-recorded and eligibility verifiability.

CHVote [126] was started in 2016 by the Swiss state of Geneva in cooperation with the Bern University of Applied Sciences. The goal was to introduce a cryptographic REV system with transparency and full verifiability that can be used in public elections. Due to financial reasons, the cooperation came to an end, and Bern University of Applied Science continued the project in 2019 with a new funding partner [126]. Their requirements for the project are E2E-V, IV, eligibility verifiability, and the distribution of trust (multi-authority) [126]. Similar to CGS97, CHVote leverages DKG among multiple election authorities. The combined election public key is then used to encrypt voters' bal-

lots. However, after the voting phase, diverging from CGS97, the votes are mixed in a RMN, and the final tally is decrypted by the election authorities.

To ensure E2E-V, the protocol uses multiple codes for verification, voting, confirmation, finalization, and abstention, which are printed by a printing authority and sent via postal mail to the voter. The voter can then use the codes to check Cast-as-Intended, Recorded-as-Cast, and Tallied-as-Recorded [126]. As with every voting system, E2E-V only holds if some trust assumptions are upheld, which for this scheme mainly means the printing authority needs to remain honest, since they can see all the codes on the voting paper and, therefore, behave as valid voters. As CHVote uses a mixnet implementation, many different election styles can be used with the protocol. The authors cover all election styles in Switzerland, like a referendum, approval voting, or cumulative voting.

4.1.6 Key Observations

This section provided an analysis of the state-of-the-art of REV research. The results indicate that, while more decentralized REV systems are proposed, many fail to provide a good set of properties required for real-world deployments. A key underlying issue is the public availability of the source code, which is crucial to establish trust in the respective system [178]. Another common issue is the scalability, especially for REV systems based on public permissionless BCs, which often present a significant performance bottleneck in the system. Moreover, REV systems based on public permissionless BCs with a PoW consensus algorithm can be costly and incalculable, as the cost of operation depends on the conversion rate of the BC's native token in exchange for fiat currency, rendering cost estimations challenging.

Furthermore, due to the uncontrolled environment of REV systems, achieving CR has shown to be a significant challenge. Achieving CR is still an open challenge for most REV systems, as some approaches to CR are sometimes incompatible with the legislation. Regarding core protocol mechanisms, most of the systems discussed use HE for this purpose. Consequently, EP and UP are not achieved in these systems, as HE depends on computational intractability. Regarding IdM, REV systems based on decentralized IdM (*e.g.*, [12]) are still sparse, despite the fact that this approach would help further distribute trust. This is because governments and the industry have not fully adopted these new IdM principles.

More research will delve deeper into EP and UP topics, as most REV systems currently do not consider them. These properties will progressively gain importance due to the progress in the development of quantum computers and the increased available computing power in general. Another area of future research could be the integration of decentralized IdM (*e.g.*, SSI) into decentralized REV systems.

Furthermore, the is one of the most important building blocks of REV systems, how it is implemented can have laege implications on other aspects, such as scalability. With decentralized PBBs, this is especially important. Amongst the REV systems inspected, three different types of configurations could be found (*cf.*, Table 4.2). The first category of REV system uses a public permissioned BC (*i.e.*, [47, 215, 309, 315]. Another group consists of REV systems using public permissioned BCs to implement their PBBs (*e.g.*, [173, 179]). Finally, some systems used private permissioned BCs for this purpose (*i.e.*, [279, 286, 329]). Note that Vocdoni [315] is mentioned in two categories simultaneously. This is because it uses the Ethereum BC as well as a permissioned BC, where Ethereum stores the final results of an election, and the permissioned BC stores the encrypted votes.

	Public	**Private**
Permissionless	Verify-Your-Vote [47] BBB-Voting [309] Open Vote Network [215] Vocdoni [315]	*none*
Permissioned	Æternum [173] Provotum 2.0 [179] Provotum 3.0 RF [148] Provotum 3.0 CAI [211] Provotum 3.0 IdM [12] Provotum 3.0 MN [172] Vocdoni [315]	Chaintegrity [329] TIVI [279] Voatz [286]

Table 4.2: Comparison of PBBs in Terms of BC / DL Type

In the end, designing and implementing a REV system for use in nationwide elections requires great prudence concerning privacy and verifiability considerations, as failure to achieve these properties will result in a loss of trust in REV systems. This thesis addresses all the issues mentioned above, focusing on the design and implementation of privacy-preserving and decentralized REV systems.

4.2 Review of Remote Postal Voting Schemes

This chapter elaborates on related work proposing protocols to enable verifiability in RPV systems, and an overview is provided in Table 4.3. Furthermore, these works are compared and analyzed with regards to the feasibility of application in Swiss RPV processes. This section summarizes the suit-

Table 4.3: Comparison of RPV Systems, ● = Provides Property, ○ = Does Not Provide Property, ◐ = Partially Provides Property, ○ = Deduced property from reference

	Ballot Privacy	Receipt Freeness	Everlasting Privacy	Individual Verifiability	Universal Verifiability	Cast-as-Intended	Recorded-as-Cast	Tallied-as-Recorded	Eligibility Verifiability	Voting Protocol
Verifiable Postal Voting [24]	●	○	○	●	●	●	○	○	●	HE/MN, DS, NIZKP, (CS)
Towards Verifiable Remote Postal Voting with Paper Assurance [217]	●	◐	○	○	●	●	●	○	●	HE, MN, NIZKP, MAC, CS
STROBE-Voting [22]	●	●	○	○	○	○	○	○	●*	HE, NIZKP
RemoteVote [66]	●	●	●	○	○	●	●	●	●	HE, CS, MN
SAFE Vote [66]	●	●	●	○	○	●	●	●	●	HE, CS, MN

ability of the protocols presented regarding their use in the context of Swiss RPV and compares the protocols by highlighting their properties and cryptographic primitives.

The protocols presented use different trust assumptions to achieve the listed properties. For instance, Benaloh et al. [24] assumes that the election procedures are public and are asserted by observers, with at least one honest observer. McMurtry et al. [217] base the privacy of their system on the assumption that, amongst others, the election authority, the postal services, and the voter's device are honest [217]. Further, STROBE-Voting [22] presumes that the ballot printers do not store secret information about the ballots they print [22]. Unfortunately, RemoteVote and SAFE Vote do not list any explicit assumptions [66].

4.2.1 Verifiable Postal Voting

In ***Verifiable Postal Voting*** [24], Benaloh et al. propose a protocol to enhance existing RPV procedures by adding cryptographic information to the Paper Ballots ( ). The protocol is based on three pieces of paper the voter prints at home: The first paper ( ) contains the voters' choice in plaintext. The second paper ( ) contains an encrypted version of the content of , together with the random values and used to encrypt both and , respectively. Finally, the third piece of paper ( ) contains a re-encrypted version of alongside a digital signature of the voter. The voter puts these three papers into the envelope and sends it to the municipality. Note that the voter keeps a copy of , which is later used for verification [24].

At the tallying facility, officials open the envelopes and verify the signatures on . Afterwards, is physically stapled to either or . Next, the papers are sorted according to their contents, resulting in four shuffled piles (, and , and ,). Finally, from each of these piles, specific information is published to the . Thus, the contains all three papers from each voter, but only one of two links (or) to preserve privacy [24].

[24] achieves Ballot Privacy (BP) under the assumption that the tallying personnel comply with the process rules defined in the protocol. However, the system does not achieve Receipt-Freeness (RF), as the voter can prove how she voted if she keeps as a receipt. Individual (IV) and Cast-as-Intended (CaI) verifiability are achieved by letting the voter check the plaintext vote before sealing the envelope. As all plaintext votes ( ) are published to the , the protocol fulfills Universal Verifiability (UV). The protocol also achieves Counted-as-Cast (CaC) verifiability by letting the tallying observers decide for each envelope whether to check for the equality of and or and , respectively [24].

4.2.2 Verifiable Remote Postal Voting with Paper Assurance

In ***Towards Verifiable Remote Postal Voting with Paper Assurance*** [217] McMurtry et al. provide a similar approach for a verifiable RPV system. This approach also leverages cryptography to enhance the traditional approach to conducting RPV. The protocol entails four phases:

1. : To set up the election, the Voting Authority (VA) stores a double envelope with the encrypted (inner) and the plaintext (outer) Voter IDs onto the .

2. : The voter then generates two random secrets to produce two commitments. These commitments are then posted to the . The voter now chooses her vote. Afterward, a MAC is generated and published on the . The voter can now print two pieces of paper: Paper 1 contains the vote in plaintext, the inputs for the two commitments, and two ZKPs. Paper 2 consists of the Voter ID. The voter now places Paper 1 in a PBE, puts it into the VE together with Paper 2, and sends it to the tallying location via postal service.

3. : After the VE has arrived at the tallying location, the VA opens the VE and removes Paper 2, which allows them to validate the voter's identity. Then, the VA gets the encrypted Voter ID from the inner envelope on the . The encrypted Voter ID is then stapled to Paper 1 blindly. Afterwards, the VA destroys Paper 2. Before Paper 1 and the encrypted Voter ID are stored on the , both ZKPs are validated [217].

4. : The VA trustees decrypt the voters' secrets cooperatively to start the tallying process. They then validate whether the commitments are valid. Finally, the system generates a new MAC from the decrypted parameters and compares it to the original MAC as posted by the voter [217].

The protocol provides BP, assuming that an attacker does not collude with the VA, the postal service, or the voter's device. Furthermore, RF is achieved, assuming all actors follow the protocol strictly. Moreover, the protocol fulfills CaI verifiability by letting the voter verify the plaintext on Paper 1 before casting. Recorded-as-Intended (RaI) verifiability is achieved by first performing the CaI check, then validating that the Voter ID on Paper 2 is correct, followed by checking the inclusion of the Voter ID in the after the voting phase. The protocol achieves UV by verifying the proofs generated during the tallying phase and verifying the uniqueness of Voter IDs on the . The presence of an EV mechanism provided by the VA is assumed [217].

4.2.3 STROBE-Voting

STROBE-Voting [22] [22] is a protocol that achieves End-To-End Verifiability (E2E-V) in both postal and in-person voting.

1. **Setup**: For each eligible voter, the VA generates two ballots containing the same options. Both ballots are encrypted using the ElGamal [95] cryptosystem on a per-row basis, which means that each candidate option is represented as an encrypted vector of zeros and a one. Each encrypted vector is subsequently hashed and truncated to generate shortcodes, which serve as a reference for the voter to ensure verifiability. Next, all encrypted options on the ballot are input into a hash function, producing a ballot hash, a hash string that ensures that all encrypted options remain unaltered. Finally, the printed ballot contains the ballot hash and the shortcodes for each option [22].

2. **Voting**: When receiving the , the voter fills out one of the and keeps the other for verification. The voter memorizes the selected options' shortcodes and the ballot hash of the selected and then casts the [22].

3. **Receival**: When the VEs arrive at the VA's premises, the VA publishes encryptions, shortcodes, and the ballot hashes of the cast and the retained . For the cast, the selected shortcodes (together with the ballot hash) are also published. The retained is decrypted and published, together with the randomness factors used to encrypt it in the first place. Finally, the VA publishes multiple to ensure the correctness of the s [22].

4. **Tallying**: The process involves adding the encrypted votes homomorphically and decrypting the tally results [22].

STROBE-Voting [22] provides E2E-V by applying the following mechanisms: The voter memorizes shortcodes selected and validates against the published shortcodes for that particular ballot hash [22]. Additionally, the correctness of the cast is checked by verifying the and the 's twin [22]. Furthermore, it achieves BP by applying encryption to the s [22].

4.2.4 RemoteVote

With ***RemoteVote*** [66], Crimmins et al. presents a work inspired by STROBE-Voting [22]. The (*cf.*, Figure 4.5) consists of the options, two encrypted ballots, each defining a column of shortcodes for all options, and a ballot ID that combines the identifiers of each encrypted ballot [66].

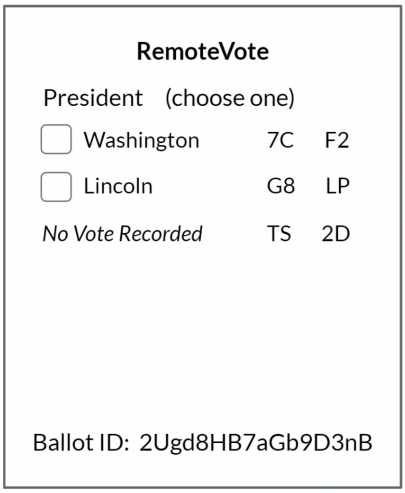


Figure 4.5: Example of PB in RemoteVote [66]

1. **Election Setup**: First, the VA constructs the ballots according to the requirements of the specific election, which involves encrypting the options together with the corresponding commitments and generating shortcodes and ballot codes [66].

2. **Paper Ballot Creation**: For each voter, the VA combines two encrypted ballots into one physical [66], which follows the structure described above (and in Figure 4.5). Note that the ballot ID, the longcodes of the encrypted ballots, and the shortcodes, together with the generated commitments, are published on the , allowing for later verification. The s are then sent to the voters by postal mail [66].

3. **Vote Casting**: The voter selects the candidates on the by filling in the ballot by hand while memorizing selected candidates' shortcodes, together with the ballot ID, as these attributes allow the voter to perform verification later [66].

4. **Receival**: When the VA receives the s, it posts the s' evidence (ballot ID and selected shortcodes) to the . From this point, voters can verify whether their vote has been recorded correctly [66].

5. **Tallying**: A MN anonymizes the commitments before calculating the final tally by revealing the committed values [66].

Identical to STROBE-Voting [22], RemoteVote [66] achieves E2E-V by allowing the voter to verify that their shortcodes have been processed correctly. From a privacy standpoint, RemoteVote [66] achieves BP and RF. Additionally, it fulfills Everlasting Privacy (EP), as it uses commitments for the tallying process instead of HE [66].

4.2.5 SAFE Vote

SAFE Vote [66] can be regarded as a sibling protocol to RemoteVote [66], which uses steps that are identical to steps 1, 4, and 5. However, the (*cf.*, Figure 4.6) is slightly different, as it contains one row of shortcodes, a scratch-off label hiding a random value, the ballot ID, and a QR-Code [66].

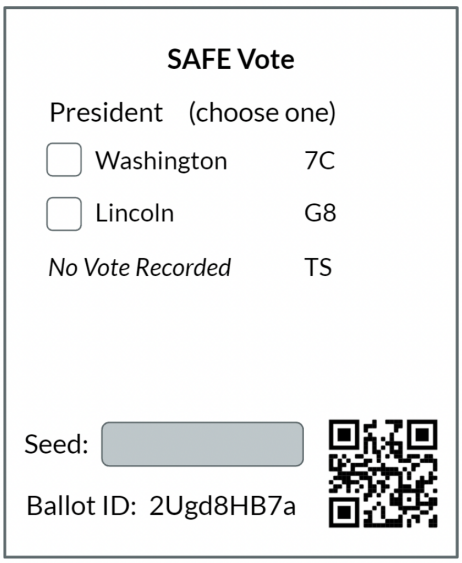


Figure 4.6: Example of PB in SAFE Vote [66]

1. **Election Setup**: Identical to RemoteVote [66]
2. **Paper Ballot Creation**: The VA produces the by printing a QR-Code containing the ballot commitment onto the and the random value used to produce the commitment as a scratch-off label [66].
3. **Vote Casting**: Before the voter selects her candidates, she can verify that the shortcodes printed on the ballot are valid by removing the scratch-off label and scanning the QR-Code [66]. With these two items, the voter can reconstruct the ballot and check whether the reconstructed ballot matches the actual [66].
4. **Receival**: Identical to RemoteVote [66]
5. **Tallying**: Identical to RemoteVote [66]

SAFE Vote [66] fulfills the same properties as RemoteVote while providing individual CaI verification compared to the collective CaI verification in RemoteVote.

4.2.6 Key Observations

This section provided an analysis of the state-of-the-art of RPV research. The analysis indicates that proposed RPV schemes are based on findings stemming from REV research, *i.e.*, the cryptographic methods applied to create E2E-verifiable voting schemes. Focusing on the applicability to the Swiss RPV, a desirable property of an RPV protocol for use in Switzerland is compatibility with existing voting procedures. Thus, a protocol should be compatible with ballot delivery and casting by postal mail. Likewise, such a protocol should not interfere with other voting channels, such as in-person voting. Furthermore, the goal is to achieve as much verifiability (*e.g.*, E2E-V) as possible while keeping the protocol complexity as low as possible, as introducing complexity into a system can lead to security issues [216], which is undesirable in a voting system. Moreover, a general requirement is the ease of use for voters and VA alike when running the protocol. A system with poor usability could negatively impact voter turnout, which is also undesirable. Last but not least, a RPV system should remain operable despite the failure of computer systems. This resilience could be achieved by basing the system on the existing RPV procedures, which can serve as a fallback.

[24] and [217] do not provide E2E-V and are not designed for postal ballot delivery. The three remaining protocols [22, 66] are compatible with postal ballot delivery and ballot casting while providing E2E-V. Furthermore, they do not impede other voting channels. When comparing the cryptographic primitives used, STROBE-Voting [22] uses HE and resulting in relatively low complexity. RemoteVote [66] and SAFE Vote [66] use HE, CS, and a MN. From a privacy standpoint,

STROBE-Voting [22] trades EP in favor of a more straightforward design while still achieving a similar degree of verifiability compared to RemoteVote and SAFE Vote.

The design specified in Section 6.2 is based on STROBE-Voting [22], as it provides a suitable combination of privacy and verifiability with a relatively simple design. Further, STROBE-Voting fits nicely into the context of Swiss RPV, requiring only minor changes to existing voting procedures which are detailed in Chapter 6. Moreover, the design of STROBE-Voting [22] is easily extendable with ideas from RemoteVote [66] or SAFE Vote [66], as both are based on STROBE-Voting [22].

Simplicity is prerequisite for reliability.

Edsger W. Dijkstra

5 Swiss Remote Postal Voting Process

In Switzerland, the federal government defined strategies to enable the use of digital technologies for public authorities and processes, including Electronic Voting (EV) [83, 291]. Private companies collaborate with Swiss authorities to actively define standards across e-Government processes [314].

As outlined in Section 1.2, the key question is: what exactly does such a minimal level of security involve? Which security metrics and mechanisms are mandatory? In the public's perception, EV often provokes a fear of change, presuming the current RPV system to be mostly analog and tamper-proof. However, it can be argued that the current Swiss RPV system is already partially EV, since many steps already involve distributed electronic systems. Thus, defining and comparing the security properties of a REV system also requires an analysis of the current RPV system in Switzerland.

This chapter is based on the publication [180], which was also presented at a security conference in Switzerland [174]. The following sections summarize major related work and terminology to formalize the Postal Voting Process Flow (PVPF) in Switzerland. The approach taken formalizes the PVPF in a step-based model, for which major assumptions made, such as trust, people involved, and technology applied, are made explicit, if known. The interpretation of social trust assumptions is discussed within Sec. 5.2, along with the risk analysis, weaknesses, and strengths of a person-based

RPV approach. Finally, an overall risk assessment is provided in Sec. 5.3, providing the basis for discussions of security-relevant comparisons to REV, while Sec. 5.4 draws main conclusions.

5.1 Legal Background

Switzerland is organized as a decentralized system of municipal and cantonal entities, working together under the umbrella of the federal government. The federal structure is also mirrored in the legal framework (*cf.*, Figure 5.1). At the root is the Federal Constitution of the Swiss Confederation, wherein Art. 39 [74] forms the basis for the Federal Act on Political Rights (BPR) [76]. In turn, Art. 91 BPR [76] is the foundation for the Federal Decree for Political Rights (VPR) [77]. On a cantonal level, the VPR builds the foundation for the Cantonal Decrees (*e.g.*, for the canton of Aargau [75]). Every canton is an independent legal entity and defines its own constitution on the basis of the Federal Constitution.

The political system is under the authority of the cantons, *i.e.*, cantonal laws and ordinances regarding political rights define elements for these processes. Various aspects of those elements are relevant to the RPV system in Switzerland, and each canton has its own decrees regarding political rights. The federal structure is mirrored down to the municipal level: each municipality decides on certain processes, again, aligned to cantonal laws and decrees. For instance, keeping a record of the electoral register is under the authority of municipalities, leading to different approaches.

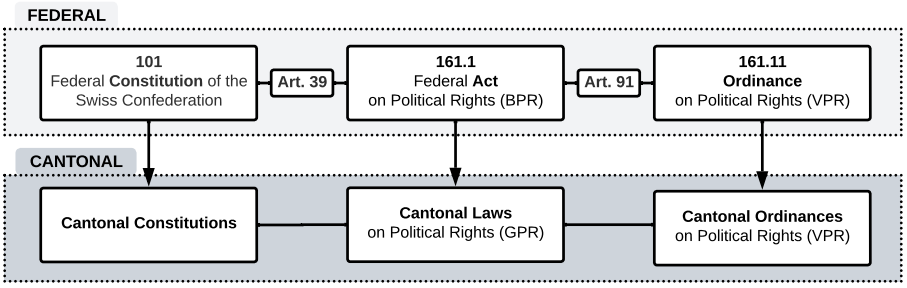


Figure 5.1: Swiss Legal Framework

A direct comparison of the Swiss RPV system to REV was performed in [273]. Other countries have discussed the usage of RPV critically because the secrecy of the ballot cannot be fully ensured [273]. From a practical standpoint, thorough documentation is the easiest way to achieve verifiability for RPV. Supervisory bodies and authorities should check the documentation and verify it [273]. Also, trust is crucial for all voting methods. And the relationship between verifiability and trust is

Table 5.1: Official German Terminology with Corresponding English Translations

German	English
Zwei-Weg Abstimmungskuvert	Two-Way Voting Envelope (VE)
Abstimmungsresultat	Voting Result (VR)
Erwahrung des Abstimmungsresultates	Legally valid determination of VR
Die Schweizerische Post	The Swiss Post (SP)
Stimmkuverts	Paper Ballot Envelope (PBE)
Stimmrechtsausweis	Voting Signature Card (VSC)
Stimmregister	Electoral Register (ER)
Stimmzettel	Paper Ballot (PB)
Vertrauenswürdiger Dritter	Trusted Third Party (TTP)

neither linear nor one-dimensional. Technical measures are not sufficient to create trust, sociopsychological aspects also have to be considered carefully. An extended literature review is provided in [180] with a focus on Switzerland, but also outlining the work done in Canada [236], Estonia [188], and Australia [280].

In order to analyze the RPV system with a focus on security aspects, the US National Institute for Standards Technology (NIST) serves as reference, outlining and standardizing terminology on the "Effort, Detection and Impact levels of Threat Events" [252, 299, 300]. Past work applied such principles to a RPV system used in the United States [251]. To apply terminology consistently, Table 5.1 defines the corresponding terminology used in Swiss legislation and their English translations.

5.2 Postal Voting Process Flow (PVPF) in Switzerland

This section details the illustrated Postal Voting Process Flow (PVPF) in depth (*cf.*, Figure 5.3), containing an end-to-end process as it is currently implemented in Switzerland. The detailed sub-steps are formalized and vary between cantons and municipalities. However, the general process adheres to the federal laws and ordinances. The PVPF is divided into six main phases, from *A* to *F*, each phase containing one or multiple sub-stages from 1 to *N* (*cf.*, Figure 5.2). The ensuing subsections are structured according to the PVPF within Figure 5.3 and describe all the different steps in detail.

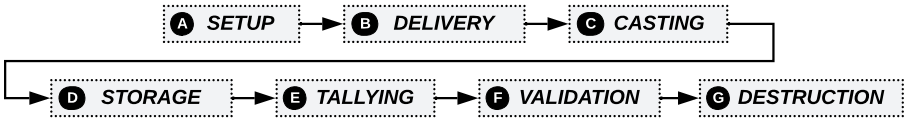


Figure 5.2: Paper Voting Process Phases

5.2.1 Setup Phase *A*

The Setup phase *A* contains four sub-stages 1-4, describing the production and assembly for dispatch of all necessary ballots and envelopes(*cf.*, Figure 5.4). The two-way Voting Envelopes (VE), the Voting Signature Cards (VSC), the Paper Ballot Envelopes (PBE), and the Paper Ballots (PB) are the physical artifacts produced in the Setup phase. The secure execution of the Setup phase is crucial, since all phases that follow rely on the sound production and assembly of those artifacts. The main stakeholders of this phase are the municipal and cantonal authorities supervising the process. Due to cost, time, and capability constraints, Trusted Third Parties (TTP) support the authorities during the Setup phase as External Suppliers (ES).

Production of voting envelopes: In Step 1, the certified two-way VEs are produced by an ES. In the canton of Aargau, the municipality secretaries place a centralized buying order [161, 266] for the two-way VEs at least a year in advance. After production, the VEs are distributed among the municipalities. In municipalities where Step 3 is outsourced, the VEs are delivered directly to the corresponding ES. The exact process steps are under municipal authority and can differ accordingly. Some cantons contract a single ES to handle the complete Setup phase *A*, mainly due to the special requirements of EV systems [82].

Production of Paper Ballots and Voting Signature Card: Step 2 consists of the production of the PBs and the VSCs. The printing of VSCs and PBs is predominantly commissioned to an ES. Each political layer in Switzerland (federal, cantonal and municipal) commissions the PBs within their legal responsibility, *i.e.*, the production of federal referendum PBs are commissioned by the federal government, cantonal PBs are commissioned by the cantonal authorities, and municipal PBs are commissioned by the municipal authorities.

A VSC contains the name and address of the eligible voter, embedded within a template customized by the municipality. It is essential that the printed credentials are valid, since the assembled VE is delivered to the credentials printed on the VSC. The voter has to sign the VSC for the ballot to be valid. A substantial amount of ballots are not counted because many VSCs remain unsigned.

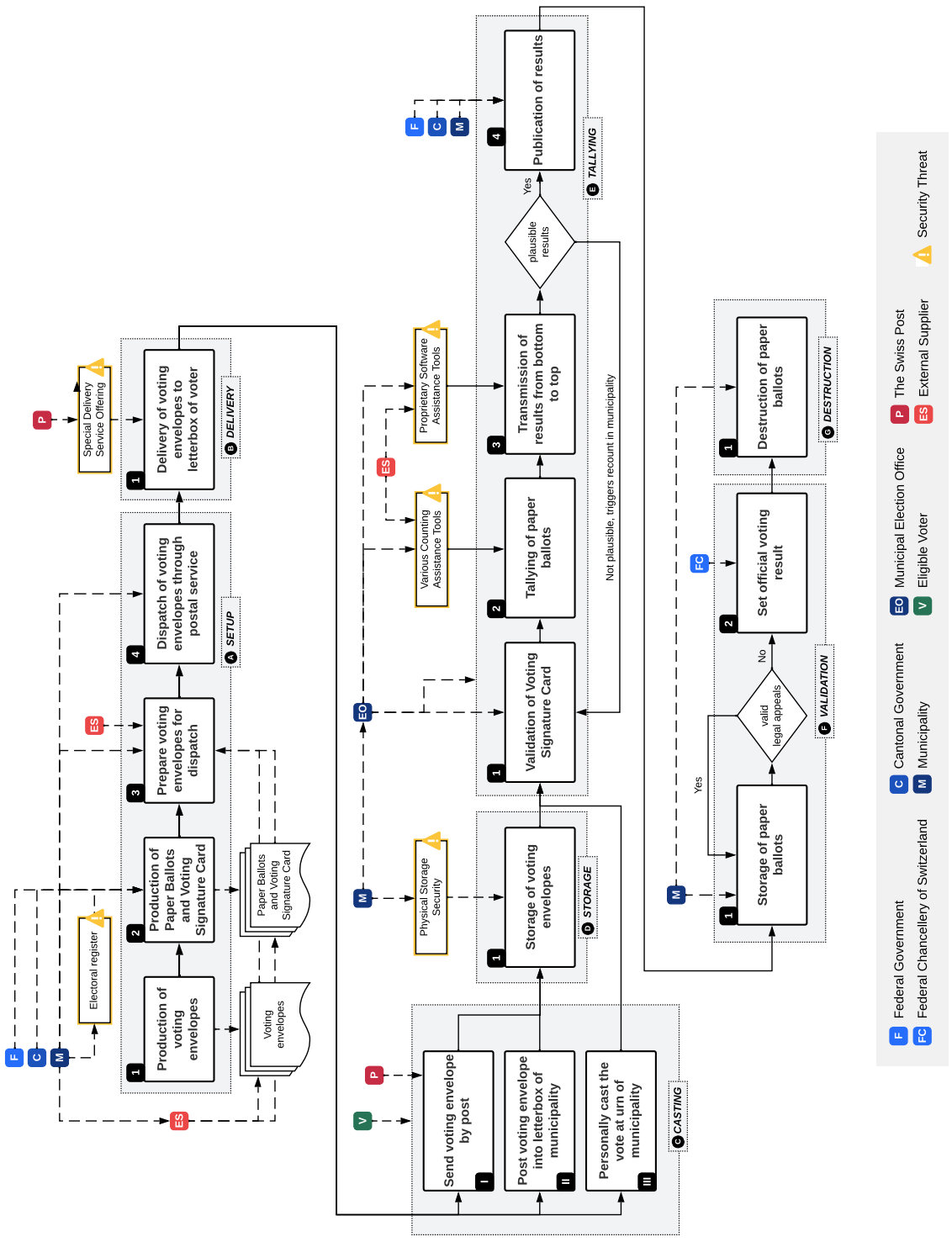


Figure 5.3: Paper Voting Process Flow (PVPF)

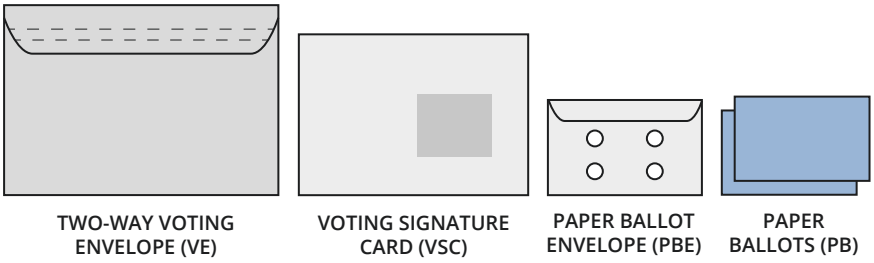


Figure 5.4: Abstract Representation of the Necessary Paper Artifacts

The individual VSCs are printed according to an electoral register (ER). The ER is a centralized register containing all eligible voters. The ER is under the authority of each municipality. Neither the cantonal, nor the federal authorities have copies of the ER [266].

Since most municipalities contract an ES for the printing of the VSCs, a dataset containing the eligible voters needs to be transferred to the corresponding ES. Most municipalities export a file from the ER (*e.g.*, XLS, CSV) and send the snapshot to the ES directly via email [265, 308]. The transmission of an unencrypted, unsigned dataset via standard email is critical, since the dataset could be tampered with (*e.g.*, fake identities created, entries being removed), either after the export, during transmission, or when the export file reached the ES. Most ERs are administered by proprietary software systems provided by companies targeting Swiss public administrations. Some cantons also provide process checklists to municipalities. For instance, the election office (EO) of the Canton of Aargau provides such checklists [161]. These checklists state that the number of VSCs should be identical to the number of eligible voters present in the ER. If issues arise, an *in-depth* manual control should take place. Whether or not to adhere to these checklists is under the authority of the municipality. Also, since printing and assembly of the VE is mostly done by ESs, the ES should verify the integrity of the printed artifacts (*e.g.*, content and amount).

Prepare voting envelope for dispatch: Step 3 concerns the final assembly of the VEs. For each eligible voter (*i.e.*, each VSC), a VE containing the VSC, PBs, and the PBE is assembled (*cf.*, Figure 5.4). The assembly is a monotonous task, often outsourced to ESs or social institutions and foundations [180]. Receiving an incomplete VE increases the possibility of the voter's abstention. According to cantonal checklists [161], handing out new PBs is only allowed if the voter can *make the loss credible*. Then, the voter's credentials should be recorded to check for attempted double voting in the Tallying phase *E* [161].

Dispatch of voting envelopes through postal service: The final step involves the dispatch of the assembled VEs. In some municipalities, the VEs are dispatched directly by the ES commissioned

for the assembly of the VEs. The Swiss Post (SP) offers a special service [87] for the dispatch and delivery of VEs.

5.2.2 Delivery Phase *B*

In Switzerland, the postal market was partially deregulated in 2009 [86]. Still, the SP maintains a monopoly on postal letters below 50 gr. Therefore, the SP is a crucial TTP, since the secure delivery to the municipality falls under the responsibility of the SP. When using the special service provided by the SP, VEs can be dispatched on a work day the week prior to the specified delivery week [87]. Then, the SP guarantees that the delivery of the VEs will take place during the specified delivery week [87].

5.2.3 Casting Phase *C*

Phase *C* outlines the three different options to cast a vote. The vast majority of ballots are not cast at the urn [121]. Statistics do not indicate whether VEs arrive through postal services (*I*) or were delivered to the letterbox by the voter (*II*).

I: The most popular way to cast the vote is to *send the VE by postal mail*. Some cantons pre-stamp the VSCs, which can then be used to return VEs free of charge [33]. For the voter, it is impossible to verify whether the ballot was successfully delivered to the municipal office. The SP offers the ability to track deliveries for an additional cost.

II: Thus, a favoured alternative is to *deliver the VE to the letterbox of the municipality*, which is then emptied by municipal officials and stored safely. According to [85], this option is still a favoured option by many voters.

III: The third option is to *personally cast the ballot at the urn*, which guarantees ballot secrecy. Casting ballots at the urn remains the most secure option to cast a vote, since the PBE (containing the ballot) is cast directly into the urn and separated from the VSC (containing the voter's credentials and signature).

5.2.4 Storage Phase *D*

Phase *D* deals with the storage of VEs that were delivered via postal service (*I*), or cast directly to the municipal letterbox (*II*). Often, an employee is tasked to retrieve the postal mail addressed to the municipal office. During votes, the VEs are collected from the SP and municipal letterbox, and then carried to the safe storage location. Past incidents describe instances where municipal employees misused that trust [34]. The storage safety varies heavily, depending on the municipality. The Federal Act for Political Rights (BPR) [76] does not specify any security requirements. Additional

considerations include the exact definition of access control for the VE storage, (*e.g.*, Who should have physical access to the VEs?). Also, defining a process for incoming VEs can increase process security (*e.g.*, How many ballots arrived at which date and time? Who got the ballots from the letterbox or post office and transported them where?).

Thus, stricter access control and a secured ballot arrival process can maximize the physical storage security. In practice, physical storage security is not prioritized, since the municipal infrastructure is often not sufficiently equipped [265, 308].

5.2.5 Tallying Phase *E*

Phase *E* specifies the process of tallying. The main stakeholders of phase *E* are the municipality and the local EO. The tallying is not regulated on a federal level and is heterogeneous among cantons and municipalities [273].

Art. 14 No. 1 BPR [76] states that every polling station should create a report containing the total number of eligible voters, the total eligible voters living abroad, the total of blank, invalid, and valid ballot papers, and the number of votes in favour and against the proposal [76]. Thus, the BPR serves as a federal guideline, without specific requirements regarding the tallying process.

Approximately 10% [241] of the ballots cast are counted with the help of Electronic Counting (e-Counting) tools, provided by ESs. The parliamentary control of the administration investigated e-Counting and concluded that the federal requirements are neither functional, nor practical, and the control mechanisms of the federal government are not sufficient [241].

Tallying of all ballots: The local EO usually hires paid and elected helpers to assist with the manual counting. In large cities, thousands of helpers are engaged to count the paper ballots [242]. The EO defines the details of the tallying process. Some municipalities use e-Counting solutions or deploy high-precision scales to weigh PBs and derive the tally from averaging the weight of (sometimes pre-counted) batches.

Transmission of results from bottom to top: According to Art. 14 No. 2 BPR [76], the cantonal government is responsible for compiling provisional results from the entire canton, notifying the Federal Chancellery (FC) of the results, and publishing the results in the Cantonal Gazette (or a special issue thereof) within 13 days of the polling day. As soon as the EO concludes tallying, the result is transmitted from the municipal EO to the cantonal EO, and from there to the FC. Some cantonal EOs deploy dedicated software systems to verify results using statistical methods. Also, most cantons make use of software provided by ESs to transmit the results. Thus, this phase also includes the use of web-based assistance tools [180].

Publication of results: The tallying phase is finalized with the publication of all results on the municipal, cantonal, and federal level. Generally, the FC publishes the collected results from the Cantonal Chancelleries in the Federal Gazette. Cantons publish the results and protocols in their cantonal gazettes. Each municipality publishes a final tally and tallying protocol with respect to the cantonal law. Mostly, the publishing process is performed by uploading documents to a public web server and displaying print-outs outside the municipal offices.

5.2.6 Validation Phase *F*

Validation: Art. 15 BPR [76] defines the validation and publication of the results. The official results can only be declared when no valid appeals are in process at the Swiss Federal Court. After that, the official result is published by the FC in the Federal Gazette and can no longer be appealed.

Storage of paper ballots: Before the results are ascertained, the counted PBs and VSCs have to be safely stored in the municipalities. It is important that the ballots remain unaltered because a recount could be triggered before the official result is determined. Past cases have shown that premature destruction of the PBs and VSCs made a full re-count impossible [26].

Determining official voting result: As soon as no more valid appeals are with the Federal Supreme Court, or as soon as a decision has been made on such an appeal, the legally valid voting result can be determined. According to Art. 15 No. 2 of the BPR [76], the validation decree shall be published in the Federal Gazette. The results being made official by the Swiss Federal chancellery finalizes the Validation phase. Since a recount is no longer possible and the result is *untouchable*, the final phase can begin.

5.2.7 Destruction Phase *G*

The final phase, *G*, involves the destruction of the stored VSCs and PBs. According to Art. 14 No. 3 BPR [76], "*following validation of the result of the vote, the ballot papers shall be destroyed.*" In practise, the destruction is usually done by physically shredding all PBs and VSCs [265].

5.3 PVPF Risk Assessment (RA)

A risk signifies the the potential impact of a threat on an information system and the likelihood of that threat occurring [57, 300]. Therefore, a risk assessment (RA) serves as the identification and determination of the impact of vulnerabilities that an adversary can exploit. A threat covers any event with a potentially adverse impact on the assessed process [252]. With respect to the RPV in Switzerland,

threat sources are groups or individuals who could feasibly attack the RPV system. Threat sources can stem from insiders or external adversaries. All Threat Events (TE) in the following RA are general in nature and require multiple co-conspiring hostile individuals or groups to achieve a large-scale effect. The effort for each threat defines the relative level of difficulty of performing a successful attack based on a threat [251]. Three relative levels of effort are defined as:

- **Low** ( ): An attack requires little / no resources or detailed knowledge of the system.
- **Moderate** ( ): An attack requires significant resources (or the ability to obtain these resources) or knowledge of the system. Inside attacks involving a small number of co-conspirators fall into this category.
- **High** ( ): An attack requires extreme resources, in-depth knowledge of the system, or even access to the systems. It also requires specific tactics, techniques, and procedures [252]. Insider attacks involving a large number of co-conspirators fall into this category [251]

Detection describes the relative level of difficulty to notice whether a particular threat has been executed in an attack [251]. Thus, attacks are more severe when they remain undetected. Three estimated levels of likelihood of detection exist [251]:

- **Low** ( ): An attack is unlikely to be detected without extraordinary resources.
- **Moderate** ( ): An attack may be detectable, but could require a large amount of resources and time. Such attacks are unlikely to be detected during the election.
- **High** ( ): An attack would most likely be detected, given proper monitoring.

The impact on PVPF was analyzed according to [252] with a focus on confidentiality (C), integrity (I), and availability (A) as defined in [299]. Some TEs - all are shown in Tab. 5.2 - are interdependent or can be combined as depicted in Figure 5.5. The following RA serves as a major discussion of potential TEs that can lead to a loss of voter confidence. The mitigation of the identified TEs concerns establishing trust and confidence in a system.

5.3.1 RA of Setup Phase *A*

The Setup Phase *A* produces all the necessary artifacts for the secure execution of the whole PVPF.

Table 5.2: Threat Events on the Swiss Postal Voting Process Flow (PVPF)

Phase	TE	Description	Effort	Detection	Impact
A	**TE1**	Delay production of physical artifacts			*A*
A	**TE2**	ER master records			*I*
A	**TE3**	ER data snapshot			*I*
A	**TE4**	Forge physical artifacts			*I*
A	**TE5**	Steal assembled VEs before dispatch			*I*
B	**TE6**	Re-route VEs	*unknown*	*unknown*	*A*
B	**TE7**	Steal VEs from voter letterboxes			*A*
C	**TE8**	Steal VEs from municipal letterbox			*I*
C	**TE9**	Re-route VEs			*C*
C	**TE10**	Cast stolen or forged VEs			*I*
D	**TE11**	Access stored VEs			*I*
E	**TE12**	Manipulate tallying			*I*
E	**TE13**	Manipulate final tally			*I*
F	**TE14**	Initiate premature destruction			*I*
G	*no major threat events identified*				

TE1 describes the *malicious delay* of $A1$ and $A2$ in the PVPF. For instance, delaying the production can be achieved by targeting contracted ESs or attacking the municipal information systems directly.

TE2 describes *tampering with the ER master records*. Often, the ER is provided and deployed by an ES. A targeted attack on an ES provider or municipal information systems with access to the ERs creates the ability to tamper with ER master records. The modification of master records can damage the integrity of ER data and the exported subset of eligible voters.

TE3 describes *tampering with ER snapshot data*. Instead of modifying the ER master records, the snapshot used to print the VSCs can be modified. When the snapshot is neither digitally signed nor encrypted, an adversary could modify the data before, during, or after transmission to the ES.

TE4 describes the *forgery of physical artifacts* with (stolen) digital templates. If an adversary gains access to digital templates used to produce the physical artifacts, the adversary can forge VSCs and PBs. Additional information may be necessary to obtain (*e.g.*, weight and type of paper used). PBEs and VEs may also be forged, stolen, or even ordered from an ES. Since most municipalities do not perform a validation of incoming VSCs (by comparing the list of eligible

voters with incoming VSCs), the attack can remain undetected. Executing TE4 requires high effort and specific knowledge of the PVPF down to the municipal level.

TE5 describes the *physical theft of the assembled VEs*. By stealing assembled VEs, the adversary can either destroy or cast ballots. The detection of this threat event relies on individual voters noticing that they did not receive their VEs, *i.e.*, the detection probability increases with every voter notifying municipal authorities.

The integrity and availability of ER is crucial for the Swiss RPV. By targeting ERs, substantial damage can be inflicted on data integrity, and also on trust in local authorities, which can undermine voters' confidence. Requirements for EV systems can serve as a reference for process improvements [82].

5.3.2 RA of Delivery Phase *B*

The Delivery phase *B* is a black box. The internal processes of the Swiss Post (SP) are not publicly available. When using the dedicated SP service to dispatch and deliver VEs, the VSC design must adhere to special layout rules to facilitate automatic batch processing [87]. The special layout of VEs could simplify identification of VEs, but requires an adversary to achieve partial control of the SP routing system. To achieve such control, a hostile individual or group can create an Advanced Persistent Threat (APT) within the SP and from there, *e.g.*, identify VEs according to specific attributes and re-route identified VEs, or attempt to delay the delivery deliberately.

TE6 describes the *re-routing of VEs*. This TE requires adversarial access to internal SP systems and the capability to covertly manipulate the postal routing. Re-routing may require a co-conspiring postal employee because re-routing a large number of VEs could raise suspicion. Assuming successful re-routing of VEs, the adversary is offered multiple options: either destroy the VEs, or open, modify, and re-cast them (*cf.*, Figure 5.5).

TE7 describes the *theft of VEs from voter letterboxes* before successful retrieval by the recipient voter. In contrast to TE5, TE7 requires the adversary to steal from individual letterboxes, not only from a single location. Similar to TE5, detection increases with every voter noticing the absence of VEs.

Phase *B* is characterized by the trust placed in one large entity, the SP. Thus, the effort and detection probability of TE6 can only be analyzed with additional information or access to internal SP systems, operations, and processes. Generally, however, an insider can achieve low detection with moderate effort.

5.3.3 RA of Casting Phase *C*

TE8 describes the *theft of VEs from the municipal letterbox*. As shown in Figure 5.5, stolen VEs can either be destroyed or opened and modified.

TE9 describes the *re-routing of VEs (before delivery to the municipality)*. Similar to TE8, re-routing offers two different options: either the adversary can decide to destroy the VEs, or open, modify, and re-cast. Similar to TE6, a co-conspiring postal employee is crucial, since delivering a large amount of VEs to a different location may alarm an honest employee.

TE10 describes the *casting of stolen or forged VEs*. An adversary can attempt to cast stolen and modified or forged artifacts to influence the voting result. The interdependence among TEs is visualized in Figure 5.5.

In official logs provided by the municipalities, there is no differentiation between channels *I* and *II*, both count as delivered by the SP. Even though 90% of votes are cast through *I* and *II*, keeping *III* remains crucial: multiple channels strengthen confidence in results because it enables cross-channel comparison with statistical methods.

5.3.4 RA of Storage Phase *D*

TE11 describes TEs originating from *physical storage security*. Depending on the municipality, one or *N* employees have access to the cast VEs. The access to VEs offers similar options as presented in Figure 5.5. Since most municipalities do not log the number of incoming VEs, the destruction of VEs can remain undetected.

Physical access to the stored ballots allows an adversary to either destroy VEs, modify them, or open VEs and break ballot secrecy. As past incidents show [34], access control to the stored VEs is again a question of trust.

5.3.5 RA of Tallying Phase *E*

TE12 describes the *risk of manipulation during tallying*. According to [241], over 10% of ballots cast in Switzerland are counted electronically. In 2014, sample checks identified errors in these counting mechanisms and concluded that e-Counting is neither more exact nor more secure than manual counting [241]. The manipulation of e-Counting requires an adversary to write targeted malware to influence the counting mechanism in his favor.

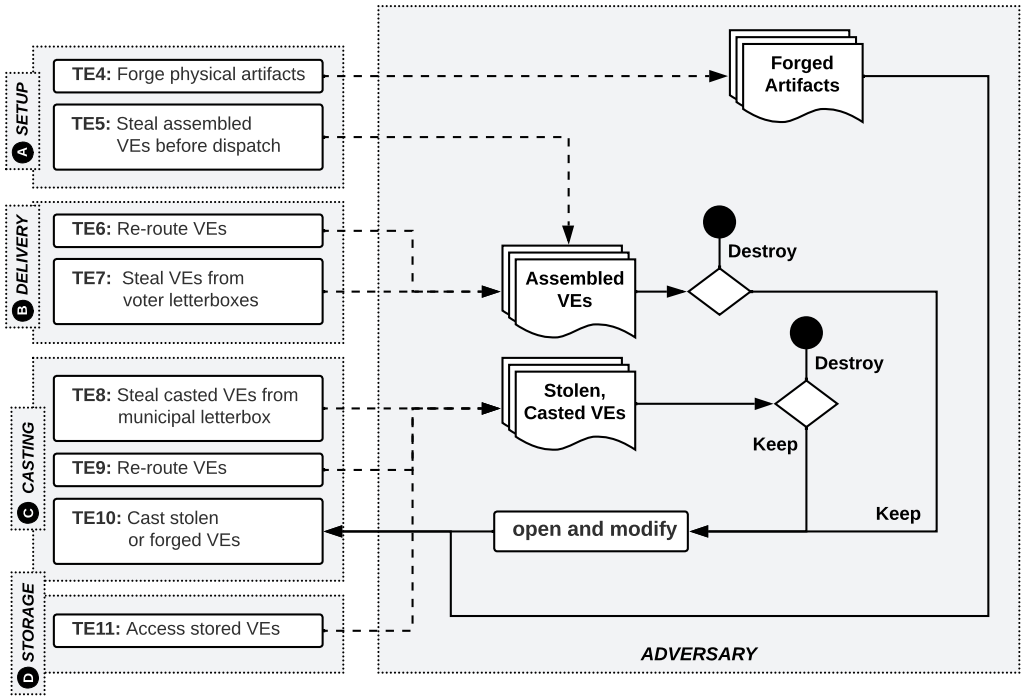


Figure 5.5: Threat Event Interdependencies

TE13 describes the possible *manipulation of the final tally*. Some cantons use proprietary software to handle vote transmission from municipalities to the cantonal EO [180].An adversary with access to these tools can tamper with the final tally. Since the manual tallying process produces logs published on a municipal level, large discrepancies can be detected by attentive observers. However, a sophisticated adversary can anticipate that and tamper with all digital traces to further obfuscate detection. Hence, the risk increases when PBs were exclusively counted electronically, without any manual counting redundancy.

The tallying phase *E* builds on the integrity of each and every individual member of the municipal Election Offices (EO). The distribution of trust builds the cornerstone of the Swiss RPV system.

5.3.6 RA of Validation Phase *F* and Destruction Phase *G*

TE14 describes the *prematurely initiated destruction* of the PBs and VSCs. The destruction of PBs and VSCs before validation by the FC makes full recounts impossible, which already occurred in 2011 [26].

Since the validation finalizes and validates the official result within Phase *G*, a recount is no longer an option. Also, PBs are now irrelevant, since legal appeals are impossible at this point.

5.4 Conclusions on the PVPF

The Swiss postal voting system is highly successful because substantial trust is placed in third parties, which includes a wide range of governmental authorities, state-owned companies, and various private companies and suppliers, as well as the individual voter. The current Remote Postal Voting (RPV) system is built on external suppliers and trusted relationships among all parties involved. For a regular citizen, the current process is hard to decipher. Thus, this thesis provides a coherent insight into the Postal Voting Process Flow (PVPF) and identifies its weaknesses and strengths with practical examples.

The main advantage of the current RPV system is its physical decentralization, but it is undercut by using centralized information systems to administer or transfer crucial data (*e.g.*, Electoral Registers (ER) or web-based assistance tools to transmit votes). Many aspects regarding the ER, assistance tools, the Voter Signature Card (VSC), or the physical storage of VEs offer room for improvements from a security perspective.

The deployment of a Remote Electronic Voting (REV) system potentially decreases the necessary amount of trust placed in institutions and people, shifting trust to verifiable processes instead [273]. As this work showed, assessing the risks of the Swiss RPV system relies on the specific process across governmental entities. This work identified crucial Threat Events (TE) and showed that the system cannot serve as a suitable reference for electoral processes [89].

Furthermore, the Swiss federal structure leads to fragmented processes across jurisdictional barriers, from federal to cantonal, down to municipal authorities. The real-world deployment of the threat events identified requires a group of hostile individuals with specific knowledge. In small municipalities, authorities and citizens are intertwined and manipulation would either be not widely effective or detected rather swiftly. In large municipalities or large cities, processes are secured. Carrying out an attack would require substantial effort from an attacker. Hence, an attack on the RPV is most likely to be successful in medium-sized municipalities, *e.g.*, where processes have not yet adapted to the larger size of the formerly smaller municipality.

Apparently, federal laws to guide the deployment of secure e-Counting tools are not yet complete. [241]. Thus, the compilation of an open and transparent list of all the electronic tools in use in the current PV flow can help to identify further threat events and enable the design of mitigation measures to better handle risks. Furthermore, the Risk Assessment (RA) needs to be extended and ultimately

applied to full real-world processes of cantons. In turn, TEs identified can be assessed in more detail and improvements can be provided to act as a comparison to EV systems.

Amateurs hack systems, professionals hack people.

Bruce Schneier

6

Decentralized Remote Postal Voting

A vital goal of this thesis is to improve the existing status quo of the Swiss RPV. In practice, security concerns have been raised around the Swiss RPV regarding voting and election manipulation by voting officials or third parties. These concerns were justified after events and instances where malicious activities were discovered in the wild [180]. To mitigate the threats described in Chapter 5, sophisticated solutions and cryptographic approaches are required: *(i)* Proverum and *(ii)* Helverify.

Proverum focuses on identity management for citizen data, which is crucial because the final tallies cannot be guaranteed without the assurance that an eligible voter cast every vote. The Proverum approach combines a private environment based on a private permissioned DL with a public environment based on a public BC. Proverum's architecture is then applied to the Swiss RPV system, successfully addressing decentralized identity management in a federalistic state.

Helverify is a full-fledged voting solution applied to the Swiss RPV, providing verifiability for RPV procedures and integrating well into the existing Swiss RPV procedures, extending the current Swiss RPV protocol with additional cryptographic procedures.

There are no dependencies among Proverumand Helverify, both solutions could be deployed independently, but both elevate the privacy and verifiability of the Swiss RPV to higher, measurable standards.

6.1 Proverum

One key property for achieving trust in voting processes is verifiability, a fundamental building block for transparency, and establishing trust in voting processes [316]. For example, public verifiability allows anyone to verify the accuracy of the voting process and the final tally independently.

Administrative verifiability plays a crucial role because verifiability mechanisms based on applied cryptography are often not easily understood by the public. From an election's administrative perspective, *administrative verifiability* is usually accomplished through an analog process with paper audit trails, enabling manual recounts. Such mechanisms rely on redundancy (*e.g.*, the four eyes principle) to mitigate collusion and minimize trust placed in single entities. According to [21], the combination of *public* and *administrative verifiability* can reach publicly verifiable voting. The integrity of an election can only be ensured when only authorized voters can cast valid ballots. *Eligibility verifiability* means that anyone can check that each ballot published on a PBB was cast by a registered, eligible voter and, at most, one vote is tallied per voter [185]. *Unforgeability* means that only eligible voters can construct authorized ballots [282]. Achieving these properties requires suitable Identity Management (IdM) to ensure the integrity of the citizen and electorate data.

From a governmental perspective, the integrity of citizen identity data is crucial for formal processes besides voting. Keeping citizen registries updated and providing identity-dependent processes (*e.g.*, the creation of Electoral Registers (ER)) is fundamental in a federalistic system, where authorities (*e.g.*, Cantons in Switzerland) retain control over their data and infrastructure. Deploying a secure IdM in a federalistic state prefers a decentralized approach while enabling secure management and a privacy-preserving exchange of identity data. Novel approaches toward a Self-Sovereign Identity (SSI) have appeared with the promise of empowering users with control over their data. SSI solutions became a suitable platform for efficient SSI solutions [323] thanks to public BCs, which offer an alternative to remedy ownership and access control issues. Users can manage credentials stored on the BC, while also improving transparency, security, verifiability, and interoperability around the IdM processes. Thus, simplified management of trusted identity information is reached, enabling government agencies to access, share, and use sensitive data while maintaining integrity.

Thus, the Proverum architecture addresses the lack of public verifiability in voting processes and introduces a new definition of Hybrid Public Verifiability (Hybrid PV).

Hybrid Public Verifiability

This work introduces a new notion of Public Verifiability: Hybrid Public Verifiability (Hybrid PV), which can be defined as *A system offers Hybrid PV when any individual can publicly verify the accuracy of*

all administrative procedures performed. The notion combines administrative and public verifiability, and thus implicitly includes the need for a private and public environment, *i.e.*, the requirement for a combination of private permissioned DLs and public permissioned BCs, clearly dividing the private audit trail (audit-trail for AV) from the verifiable public information (audit-trial for PV).

Since, by definition, public verifiability includes resistance against *conspiracies of any size*, such an assumption is practically infeasible in many use cases *e.g.*, Proverum applied to the currently deployed Swiss RPV, since many External Suppliers are trusted, as well as municipal authorities, employees, and citizens counting paper ballots [180]. On the one hand, the underlying trust model of DLs and BCs depends on the consensus of these networks *i.e.*, on the architectural choices made. *I.e.,* given a network operating with the Proof-of-Authority (PoA) consensus and n authorities, a conspiracy of up to $n \ 2 \quad 1$ authorities can be tolerated. On the other hand, a Byzantine Fault Tolerance (BFT)-style consensus merely tolerates up to $1 \ 3$ malicious nodes. Nevertheless, such an attack could easily be detected by other network participants (*i.e.*, other authorities). However, future approaches can surpass these limitations by using the Proverum architecture, achieving resistance to conspiracies of any size. Such a security property would require the deployment of additional cryptographic protocols embedded in the Swiss RPV process.

6.1.1 System Design

The key goal of Proverum is to provide a feasible approach to Hybrid PV for voting and electoral processes, which allows the public to verify data in a public environment, while maintaining a privacy-preserving and verifiable audit-trail in the private environment.

Proverum defines a practical approach, offering Hybrid PV for voting, allowing the public to verify data within a public environment, while maintaining a privacy-preserving, verifiable audit-trail within the private environment. While the Proverum design can be applied beyond voting, *e.g.*, for electoral processes or sharing information in the health-care sector (*e.g.*, reporting verifiable infection numbers during a pandemic), this paper's focus is on applying Proverum to the Swiss RPV system.

Thus, Proverum combines a public and private environment, allowing for tamper-proof audit trails, published by peers operated by authorized entities, that are independently verifiable by anyone(*cf.*, Figure 6.1). The public environment's deployment depends on the specific requirements of public verifiability, it can be set up by *(i)* publishing to a public BC, *(ii)* deploying a public permissioned BC, or *(iii)* granting partial read access to a subset of the private environment.

The private environment is formed by authorities collaborating as a private permissioned DL. Trusted authorities are authorized to participate in the DL and thus, they can be governmental author-

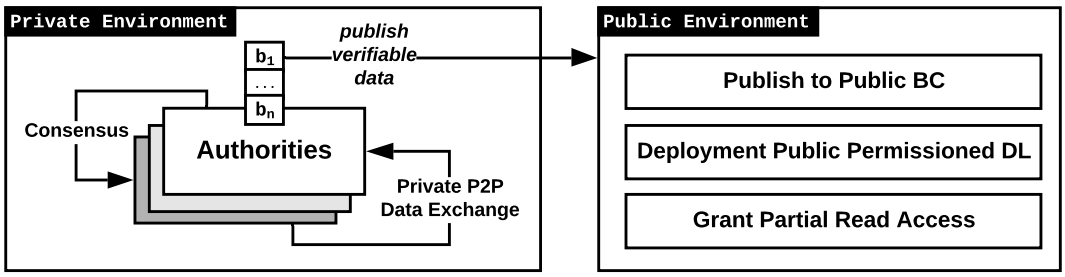


Figure 6.1: Proverum System Design for Hybrid Public Verifiability

ities or private companies. Authorities execute a consensus protocol, validate transactions, and group them into blocks while exchanging data in a Peer-to-Peer (P2P) fashion. Therefore, the private environment provides the distribution of the trust, including a privacy-preserving, verifiable audit log.

Private Environment Architecture

The private environment of Proverum requires a high-level systems architecture, depicting different authorities and their access to various Smart Contracts (SC) deployed on DLs. Since the identities of authorities maintaining the private environment are known, there is no need for a probabilistic consensus algorithm, *e.g.*, Proof-of-Work. Therefore, a suitable Byzantine Fault-Tolerant (BFT) consensus can be used. Depending on the trust model, a single authority or a set of authorities is responsible for forming the consensus.

Figure 6.2 depicts the overview of the network architecture. Network participants are cantons, municipalities, the Swiss confederation, and External Service Providers (ESP), *e.g.*, manufacturers and the Swiss Post. For fault-tolerance reasons, each participant hosts at least two peers and acts as a Certificate Authority (CA) to issue certificates.

Two purpose-specific SCs for *(i)* Citizen Management (CM) and *(ii)* Result Publication (RP) allow for keeping these SCs confidential among those involved. For instance, the ESP only has access to the *(i)* CM SC, while Swiss government authorities have access to the *(ii)* RP SC. A consortium of all authorities forms the consensus. ESPs are not part of that consortium but have read access to the External DL.

SCs and transactions persist on three different DLs. For one, there is an instance of a Federal Ledger that government entities are connected to. Second, Cantonal Ledgers include one corresponding Canton and its subordinate municipalities, including the Swiss Confederation. In a country-wide deployment, 26 Cantonal Ledgers exist. Electoral authorities join several External Ledgers for ESPs on a municipal, cantonal, and federal level. Since citizen data needs to be kept private at all

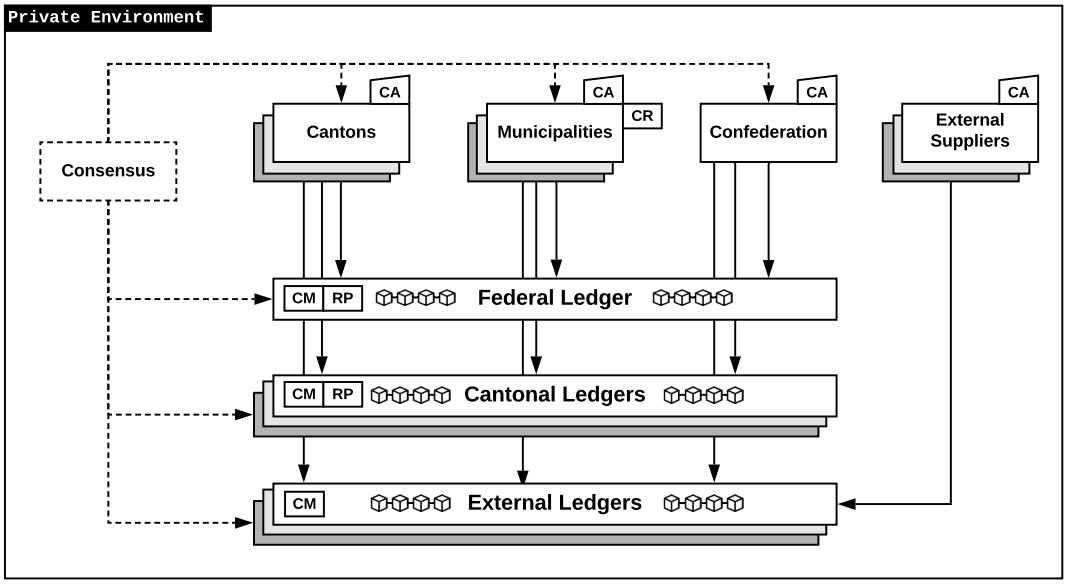


Figure 6.2: Proverum Private Environment Architecture

times, each municipality operates a private data collection maintaining the Citizen Registry (CR) without exposing any details on a shared channel.

Public Environment Architecture

The public environment of Proverum can be supported by three different designs that are not mutually exclusive, but can be combined to achieve public verifiability. For the public environment, a public permissioned DL can be applied using a Proof-of-Authority (PoA) consensus mechanism since PoA reduces proof efforts, meaning that only participants that have been given authorization can produce blocks [175]. Different trust models apply to each design, thus, depending on the specific requirements to be met, the Proverum architecture allows for all three options to be used. Each design option (summarized in Table 6.1) achieves public verifiability. With *(i)* the publication to a public BC mainly relies on the data being authentic. For *(ii)*, the deployment overhead is large, but the environment achieves the best public verifiability, showing a fully transparent audit trail that the public can audit and authorities can control. Option *(iii)* opens up the Application Programming Interface (API), is easily implemented, but also achieves the weakest notion of public verifiability since a single trusted endpoint represents a single point of failure.

1. **Publishing to a Public BC** enables the publication of data and proofs (*e.g.*, cryptographic hashes of ERs, aggregated voting results, or Non-Interactive Zero-Knowledge Proofs) by cryp-

Method	*Trust Model*	*Key Management*	*Deployment Effort*	*Public Verifiability*
(1) Publish to Public BC	Trust in *(i)* permissionless consensus, *(ii)* developers of BC, *(iii)* authorities	Keypair per public BC and per authority in private environment	Medium	To query SC
(2) Public Permissioned DL	Trust in authorities	Keypair per authority in public and private environment	High	To query full BC
(3) Grant Partial Read Access	Trust in Authorities	Keypair per authority for private environment	Low	To query API

Table 6.1: Comparison of Proverum's Public Environments

tographically signing a transaction containing this data and broadcasting it to the respective public BC. This requires appropriate key management on the side of publishing authorities, since every public BC requires the management of public/private keypairs to perform the transaction signature. Alternatively, one can rely on [267] to send and retrieve data within a transaction to multiple BCs without being locked into a single platform. With such a BC-agnostic platform, the immutability of public BCs is exploited with less complex barriers. The publication on a public permissionless BC, such as Bitcoin or Ethereum, offers transparency and public verifiability. The trust model depends on the specific public BC chosen since they vary in their selection of consensus algorithms. However, since public permissionless BCs do not rely on any Trusted Parties, the public can be assured that the data retrieved was stored in a censorship-resistant and tamper-proof manner [268].

2. **The Deployment of a Public Permissioned DL** requires additional operations. This is similar to the approach used in [175], where a Public Permissioned DL serves as a publicly readable PBB. Therefore, the trust model is comparable to the private environment, since the DL is formed by trusted authorities. The main difference compared to publication on a public BC is that authorities can censor transactions in a permissioned setting and change the content of the DL. However, since the public can read all data that persists on that DL, such changes could be observed and audited by the public, which is allowed to participate as a full node, persisting and duplicating the DL. From an economic perspective, the deployment of a public permissioned DL requires an operational effort to maintain and operate.

3. **Granting Partial Read Access** to a subset of the private environment can be offered by providing a public interface that can be queried by anyone. However, providing a centralized point of failure to a distributed system providing verifiability is not optimal, since it separates private transactions from the public, *e.g.*, by only giving access to a public interface that can be

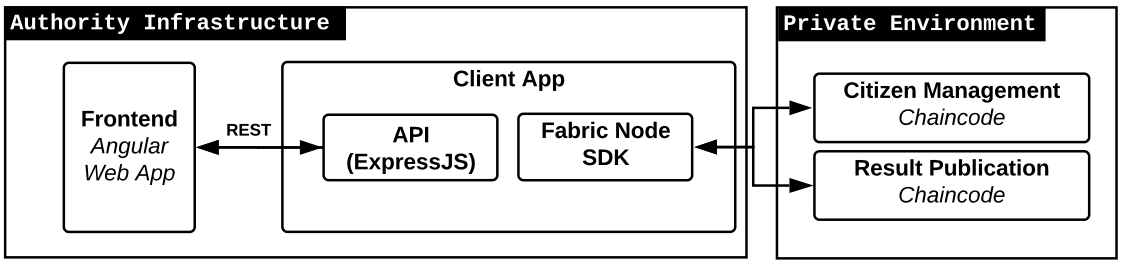


Figure 6.3: Overview of the Proverum Prototype

queried. The trust model here resembles a central server since no distribution of trust is taking place towards the public.

6.1.2 Implementation

Hyperledger Fabric (HLF) is used for the private environment instance. Therefore, SCs are referred to as *Chaincodes* and DLs among a subset of authorities are referred to as *Channels* [151]. The PoC focuses on the implementation of the private environment. The source code is available in [246].

The prototype shows two components executed on the authority's infrastructure: *(i)* a frontend web application implemented in Typescript, using Angular in a Model-View-Controller architecture [117]. The frontend serves a dashboard showing the current network status. It contains input forms and pop-up dialogs to enable authorities to submit transactions easily and trigger chaincode calls through the Client App (*cf.*, Figure 6.3). Second, *(ii)* the Client App interacts through the Fabric Node SDK with the BC infrastructure and encapsulates the various chaincode calls within REST endpoints based on the ExpressJS web application framework.

In HLF, the global state of the DL is separated based on the chaincode that accesses it. Therefore, two world states are maintained, one for the CM and one for the RP chaincode. The separation of states is required since both chaincodes are distinct and involve different stakeholders. All chaincodes are written in JavaScript for the node.js runtime environment using the HLF Chaincode Shim API to process DL states and communicate with other peers [151]. The CM chaincode contains Create, Update, and Delete functionality to manage CRs and the generation and sharing of the ERs. Additionally, the CM chaincode allows for the generation of Non-Interactive Zero-Knowledge Proofs, proving the eligibility of voters contained in the ER.

Prototype Network Setup

The prototype network is composed of the confederation, two cantons, and two municipalities per canton (*cf.*, Figure 6.4). Additionally, the Swiss Post and a single ESP (*e.g.*, a voting artifact supplier) serve as a provider for all municipalities. A total of 2,202 municipalities are served in a country-wide deployment, split up among 26 cantons, including different ESPs. Each stakeholder hosts two peers for fault tolerance purposes. HLF forms consensus by using an Orderer, a dedicated node arranging transactions in a deterministic fashion [151]. All authorities of the consortium are part of the Ordering Service (*cf.*, Figure 6.4). Thus, they can serve as Orderer and create Channels.

The CM chaincode is required on all peers, while the RP chaincode is only maintained on government peers to keep it separate from any external channels. Thus, four different channels exist in the network: the federal channel, where government authorities are connected, while cantonal channels are occupied by the respective canton and its two subordinate municipalities. Municipalities 1 to 4 and ESPs maintain private citizen data in an instance of a database.

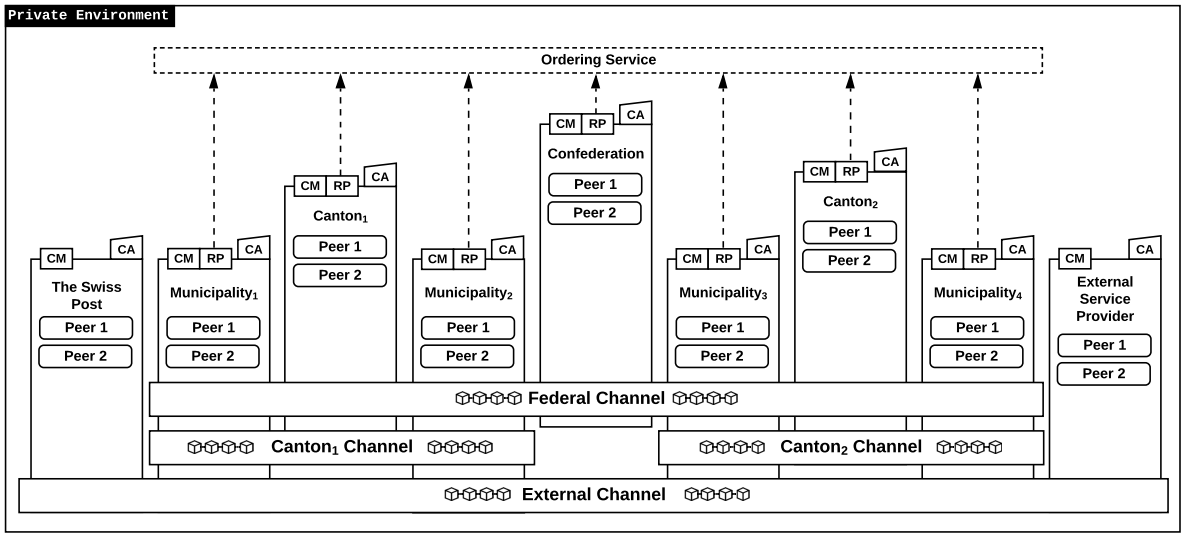


Figure 6.4: Prototype Network Architecture

Data Model

The data model is derived from the eCH eGovernment [314]. Information is encapsulated in a state object, which can be queried with a state key. Citizen states are modelled according to `eCH-0011`. These fields contain basic personal information encapsulated in data types defined by, `eCH-0044`, `eCH-0045`, and are depicted in Figure 6.5. The citizen-state contains a method to invoke a voting

restriction in case a person becomes ineligible to vote. Citizen states may have a corresponding voting citizen-state counterpart, whether the citizen is an eligible voter, implying Swiss citizenship, is an adult, a main residency in the registered municipality, and has no voting restrictions. Therefore, eligible voters show an aggregation relationship to the voting list state, representing the ER. These states are all private, thus they are off-chain, and persisted in the municipality's private data storage. Only the corresponding cryptographic hash digest of individual eligible voters, the ER, and plain voting results are persisted on-chain.

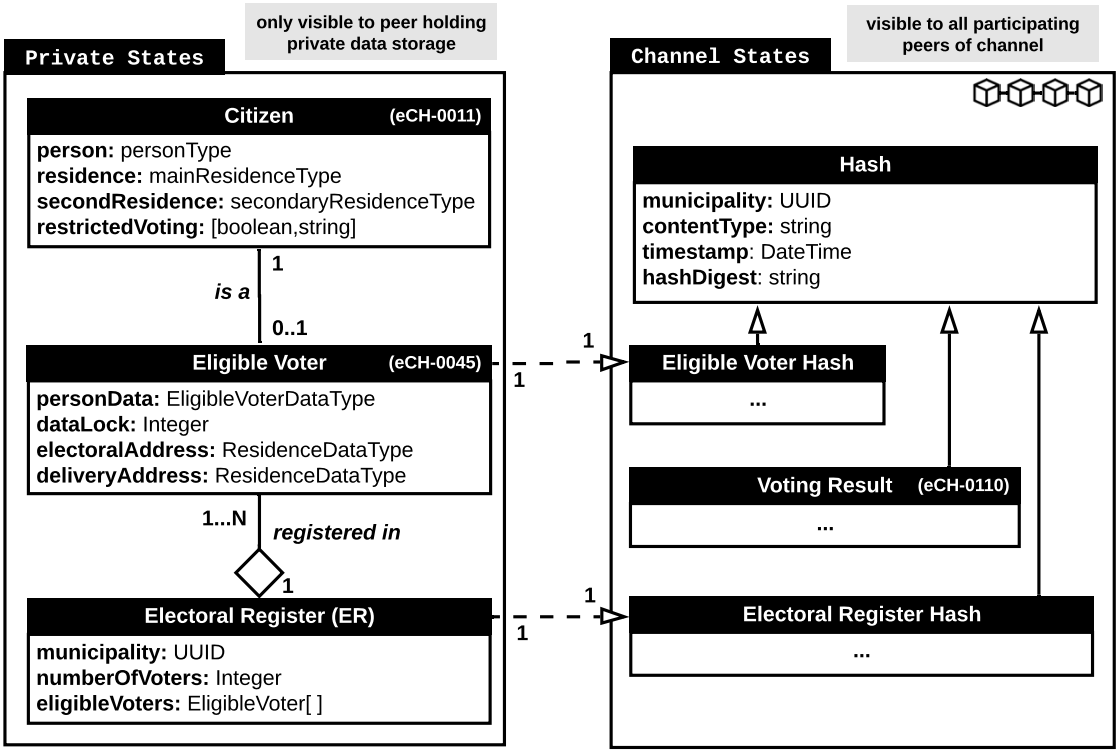


Figure 6.5: Class Diagram

6.1.3 Use Case Evaluation

Three Use Cases (UC) are defined to evaluate the Proverum prototype. Each UC shows different operations and actions being performed by the stakeholders involved. *UC1* focuses on the IdM of citizens, specifically, the relocation of citizens. *UC2* and *UC3* focus on the support for voting and electoral processes in the Swiss RPV, especially the generation and validation of ERs and the voting and election result publication process, respectively.

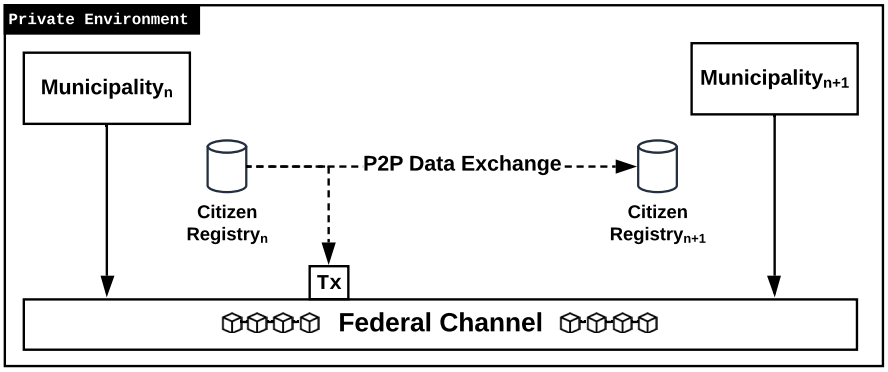


Figure 6.6: *UC1:* Citizen Relocation

UC1 Citizen Relocation:

In the event of a citizen relocating from municipality n to municipality n 1, citizen data needs to be *(i)* sent first from n to n 1. Then *(ii)* the original municipality n deletes all records. Since citizen data needs to remain private to these municipalities, they are shared P2P via a gossip protocol (*cf.*, Figure 6.6). The Ordering Service is not involved and cannot access private data, only peers of authorized municipalities have access. Thus, private citizen data is not included in the transaction to the Orderer [151]. Only a cryptographic hash digest of the private data is endorsed, ordered, and persisted on DLs of peers on the Federal Channel. Finally, the cryptographic hash serves as evidence of the transaction and is used for state validation and can also be used for audit purposes.

UC2 Generation and Validation of Electoral Registers:

As depicted in Figure 6.7, when a new vote is scheduled, municipalities generate the respective ER and the subset of the CR containing all eligible voters. The CM chaincode manages the generation of the ER. The CR is checked and excludes all citizens that *(i)* do not possess Swiss citizenship, *(ii)* are not older than eighteen, or *(iii)* do not have their primary residence within the given municipality.

The ER is exchanged with either one or multiple ESPs, which will prepare and print physical voting artifacts (*e.g.*, voting envelope, paper ballots, and paper ballot envelope). Additionally, the municipality publishes cryptographic hashes of the complete ER and hashes of each voter on the external channel, serving as time-stamped proof for verification or audits. When the manufacturer receives this private data, it will be checked against published hashes to ensure that the transmitted data is correct. This allows the Swiss Post (SP) to verify physical artifacts upon receipt. Specifically, as soon as the SP receives them, each envelope is scanned to verify whether the hash of each ballot is con-

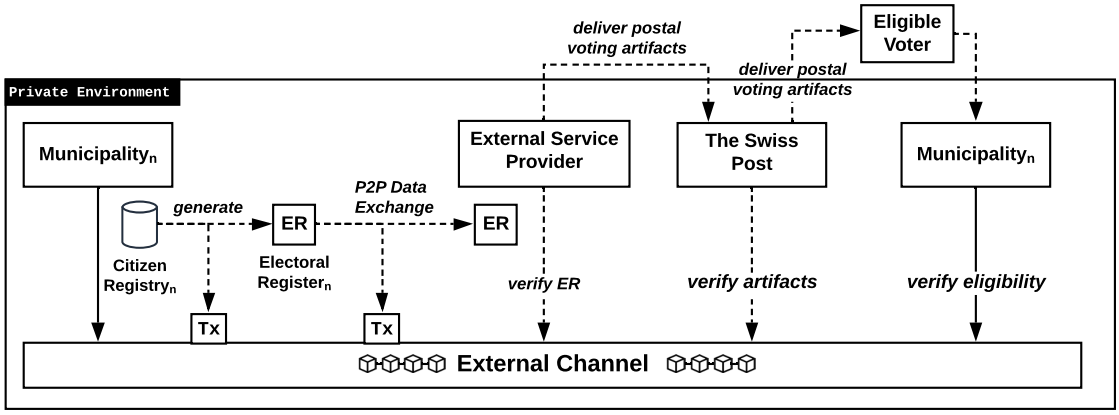


Figure 6.7: *UC2:* Generation and Validation of Electoral Registers (ER)

tained in the ER, also verifying the completeness of the batch by computing the hash of the complete list. This allows for the detection of loss, theft, or incomplete production of physical voting artifacts. Voters can gain public verifiability by seeing a read operation on the External Channel through personalized hyperlinks, *e.g.*, being embedded on the postal voting artifacts as a QR (Quick Response) code.

UC$_3$ Accumulation and Publication of Preliminary Results:

Authorities often use insecure communication channels (*e.g.*, telephone, fax, or non-secured electronic mail) to exchange preliminary results within Switzerland. From a local and global perspective, election laws do not require any guarantee regarding the integrity of electronically transmitted results, which bears the potential for attacks [285]. Thus, UC$_3$ defines a secure process to ensure the integrity and auditability of preliminary and final election results.

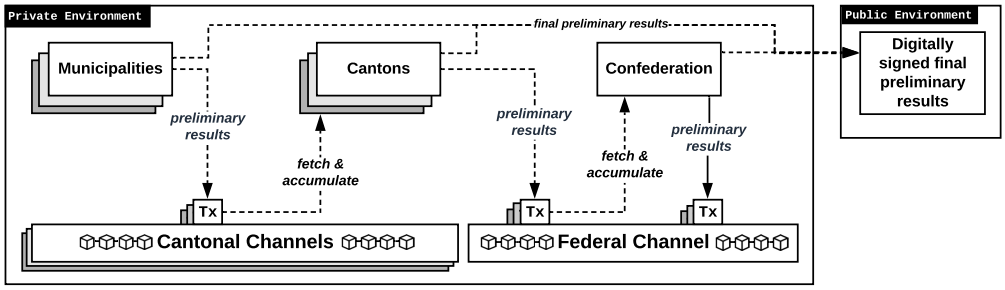


Figure 6.8: *UC3:* Secured Accumulation and Publication of Preliminary Results

Therefore, UC3 details the propagation of preliminary results from a municipality to the cantonal level by publishing results through Cantonal Channels (*cf.*, Figure 6.8). In turn, cantons perform plausibility checks, and, if necessary, trigger manual recounts. Furthermore, cantons publish accumulated results from all municipalities and publish these on the Federal Channel, where the federal government will assemble to tally the preliminary results.

Municipalities, cantons, and the confederation sign final preliminary results digitally and publish the data to the public environment, especially to a public permissioned BC or a public BC, which enables public verifiability. While still relying on the integrity of these individual results published by the different authorities involved, UC3 reaches the verifiability for the preliminary tally publication. Instead of relying on insecure communication channels, the preliminary results' authenticity and integrity are cryptographically secured.

6.1.4 Threat-based Evaluation

The evaluation of Proverum is grouped according to Threat Events (TE) for the Swiss Postal Voting Process Flow (PVPF), as outlined in [180]. Table 6.2 lists all TEs, alongside the respective Mitigation (M) techniques that are considered relevant here. Fundamental elements contributing to mitigation include: *(i)* the Public-Key Infrastructure paired with *(ii)* the use of the private Proverum environment as a distributed, immutable audit trail on a private permissioned DL, *(iii)* the public environment for public verifiability, and *(iv)* the use of SCs to automate verifiable and distributed workflows for a decentralized IdM.

TE1 refers to targeted cyberattacks on ESPs or municipal information systems (*e.g.*, in the form of Distributed Denial-of-Service attacks). A single explicit mitigation strategy does not exist, as cybersecurity best-practices need to be applied across all processes and all infrastructure involved.

TE2 refers to tampering with CR master records. As of today, municipalities and cantons manage their CR in centralized information systems. In contrast, Proverum records every change in a CR master record in a private DL's immutable records, which simplifies audits, error-detection, and provides an indication of fraudulent changes in the CR and, subsequently, in the ER.

TE3 describes the risk of ER snapshot data being manipulated in transit from the municipality to an ESP, *e.g.*, to print VSCs and assemble VEs. As outlined for UC2, the ER now persists in the Cantonal Ledger in a digitally signed transaction that persists in the immutable DL.

Table 6.2: Threat Events and Mitigation Techniques of the Swiss RPV [180]

TE	Description	Mitigation
TE1	Delay production of physical artifacts	*no Mitigation*
TE2	ER master records	*Immutable Audit Trail*
TE3	ER data snapshot	*Cryptographically signed Tx*
TE4	Forge physical artifacts	*Eligibility Verifiability Mechanism*
TE5	Steal assembled VEs before dispatch	*Artifact Verification Mechanism*
TE6	Re-route VEs	*Immutable Audit Trail*
TE7	Steal VEs from voter letterboxes	*Report, Blacklist and Re-Issuance*
TE8	Steal VEs from municipal letterbox	*Report, Blacklist and Re-Issuance*
TE9	Re-route VEs	*Immutable Audit Trail*
TE10	Cast stolen or forged VEs	*Artifact Verifiability Mechanism*
TE11	Access stored VEs	*Immutable Audit Trail*
TE12	Manipulate tallying	*Smart Contract & Audit Trail*
TE13	Manipulate final tally	*Smart Contract & Audit Trail*
TE14	Initiate premature destruction	*Cryptographically signed Tx*

TE4 describes the forgery of physical voting artifacts, which requires specific knowledge of the Swiss PVPF and access to various digital templates. With Proverum, the eligibility verifiability is ensured, since each ballot cast is cross-checked with the ER, which was published on the Cantonal Ledger (*cf.*, UC2, Figure 6.7). To consider an attack successful, the adversary requires access to the authorities to create fake identities, which would need to be included in the CR and ER in order to pass verification checks during tallying.

TE5 refers to the theft of assembled VEs before they are dispatched. As proposed within UC2, the SP verifies the receipt of all VEs, which ensures that not a single VE is missing. In case of theft, the SP will realize that they are missing and report the theft to Proverum, which automatically *(i)* triggers a re-issuance of a new ER with newly salted hashes, so the hashes are different from the ones issued previously and *(ii)* all the stolen VEs are invalidated in an explicit blacklist transaction. These blacklists are checked during the tallying phase.

TE6 describes the re-routing of VEs, which assumes an adversary has access to the SP's internal systems and may require co-conspiring postal employees. Proverum offers the immutable audit trail. Hence, the detection of such attacks is easier, since irregularities in the delivery of VEs can be detected by any member of the respective DL (*e.g.*, the Cantonal Ledger).

TE7 describes the theft of unused VEs from voters' letterboxes. Even if all prior steps of the PVPF were executed correctly, the mitigation of this attack depends on the voter noticing a theft and *(i)* reporting it to the municipality's election office, which can *(ii)* re-issue a second set

of physical voting artifacts to be used to cast a valid, eligible vote. The stolen VEs are *(iii)* invalidated in an explicit blacklist transaction and newly created physical artifacts are registered on the Cantonal Ledger.

TE8 describes the theft of VEs from the municipal letterbox, which can contain *(i)* VEs returned by SP or *(ii)* manually returned by voters. This distinction is essential, since manually returned VEs are not reported as *returned* in the shared DL. Therefore, if a voter knows he/she returned the VE and wants to verify the audit trail and the VE was not registered (and reported as *received*) by the election office, the VE appears to be missing. Similar to TE7, this needs to be *(i)* reported, *(ii)* old physical artifacts are blacklisted and *(iii)* physical artifacts are re-issued. Proverum records all actions in transactions on the DL, thus creating an audit trail.

TE9 refers to the re-routing of VEs as soon as they cast and returned via SP. This is handled the same as with TE6.

TE10 refers to the casting of stolen or forged VEs. Similar to TE4, an adversary is required to cast stolen or forged VEs and an adversary would require access to the CR.

TE11 describes the threat of an attacker gaining access to VEs cast within the respective municipality and, thus, is not a threat that can be mitigated by purely digital means since it requires trust in authorities. By contrast, Proverum addresses the risk of theft of VEs by logging and publishing all incoming VEs on the DL, thus, making the disappearance of any VE detectable.

TE12 refers to the risk of manipulation during tallying. Although the manual counting mechanisms are out of the scope of Proverum, the private environment could be used to record either *(i)* individually counted tallies or, as implemented in UC3, preliminary results. The SCs enforce the correct counting of the results entered and allow all peers in the DL to verify the correct execution, thus distributing the trust among all authorities. Further, instead of using insecure communication channels, cryptographically signed and broadcast transactions are persisted on the DL.

TE13 describes the manipulation of the final tally. Since most cantons use proprietary software to handle vote transmission for preliminary results [285], an adversary can tamper with these results. Similar to TE12, the cryptographically signed transaction ensures that results can not be tampered with, since they are included in the immutable and distributed DL.

TE14 describes the triggering of premature destruction. Municipalities await a formal message over insecure communication channels to receive the approval to destroy all physical voting arti-

facts. For instance, a potential attack could send emails to various municipalities, triggering premature destruction. Proverum mitigates this by using the DL to communicate only authenticated, and verifiable destruction commands to municipalities.

Thus, the Proverum architecture is generally an effective combination of private and public environments to achieve Hybrid PV and serves as a highly suitable approach for decentralized Identity Management (IdM) of citizens, including unforgeable states. The private environment is based on a private permissioned DL, which forms the immutable audit-trail, enables the decentralized IdM, and allows for an automated process for otherwise manual and error-prone analog processes with SCs. The public environment achieves transparency by using public (permissioned DLs) to reach public verifiability. The physical artifacts are verified again as soon as the votes are cast and returned to the municipality. Forged physical voting artifacts can be easily detected since the cryptographic hash digest generated in the first step was published on the channel and verified. However, it is still possible for an adversary to steal physical artifacts (and then cast them) when they are in transit between the Swiss Post and the Eligible Voter. However, it would be possible to detect theft when many voters suddenly ask for an additional set of voting material.

6.2 Helverify

Helverify [190] is the first end-to-end verifiable voting system PoC tailored to the Swiss RPV procedures. As outlined in Chapter 5, there have been multiple incidents involving fraudulent activities during elections or votes (*e.g.*, [13, 169, 170, 192, 193]) to manipulate the outcomes. These concerns raise the key question: *How can a voter be sure that her ballot was cast, recorded, and counted as she intended?* With the current procedures, voters can check their ballot before putting it into the voting envelope to ensure it was cast correctly. Alternatively, voters could cast their ballot at the polling station, ensuring that the ballot was cast and recorded correctly. Still, there is currently no way of verifying that a vote or election has been performed correctly, as current procedures rely on statistical analyses of the results to discover anomalies (*e.g.*, [193]). The lack of verifiability is particularly serious if voters cast their ballots by postal mail.

Consequently, the current Swiss RPV procedures are the primary influence on the requirements for Helverify (*e.g.*, regarding scalability, performance, and usability), which will effectively augment the current Swiss RPV procedures instead of replacing them, allowing said procedures to act as a fall-back in case something goes wrong. Helverify leverages the combination of a Proof-of-Authority (PoA) BC technology and a distributed file system (*i.e.*, IPFS) to provide a to store the election evidence. Helverify's architecture allows for immutability, public accessibility, and

scalability, which are crucial properties of a . Moreover, the decentralized design permits seamless integration into the current Swiss RPV procedures, which are decentralized by design [180]. Furthermore, HE, threshold cryptography, and have been applied to achieve End-To-End Verifiability (E2E-V) while preserving Ballot Privacy (BP) at the same time.

6.2.1 System Requirements

This section delves into the requirements of Helverify , implementing a verifiable Swiss RPV protocol. The protocol itself is based on the mechanisms of STROBE-Voting [22], with adaptions toward the Swiss RPV context. Furthermore, this section discusses the system architecture, which is the basis for the implementation (*cf.*, Section 6.2.4).

Before delving into concrete use cases, it is essential to determine and classify the requirements of a RPV system. The requirements for this work are mainly influenced by the current Swiss RPV procedures [180] and the current Swiss legislation [76, 80]. The following sections discuss the Functional (FR), Non-Functional Requirements (NCF), and Constraints (C) to be considered in the context of Swiss RPV.

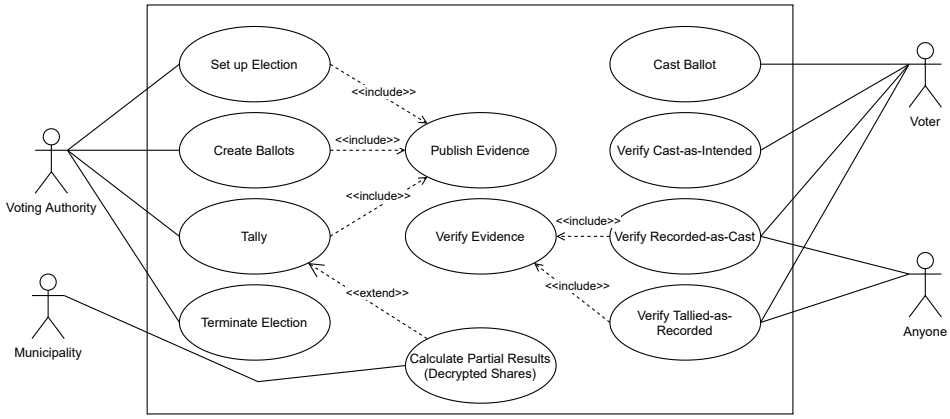


Figure 6.9: Use Cases of Helverify

Requirements and Constraints

The Functional Requirements (FR) for a verifiable Swiss RPV system are relatively straightforward, as they mainly focus on providing verifiability:

- **FR1.** The voting process *must* achieve E2E-V for the voter.
- **FR2.** The verification procedures *must* be optional for voters.

- **FR3.** The VA *must* be able to create the PBs and publish evidence of correct ballot creation.
- **FR4.** The VA *must* run the tallying process and publish evidence of correct tallying.
- **FR5.** The voter *must* be able to cast a ballot.

Quality aspects are fundamental in RPV systems, as the user base of such a system is potentially very large. Thus, the Non-Functional Requirements (NFR) primarily revolve around scalability and usability:

- **NFR1.** The system *must* scale for national elections and votes.
- **NFR2.** Verifiability mechanisms *must* be easy to use for voters and observers.
- **NFR3.** The voter's privacy *must* be protected (BP).

RPV systems must comply with various legal and procedural guidelines, which are referred to as constraints in the context of requirements engineering. Thus, to design a usable RPV system in the context of Swiss RPV, it is crucial to account for the following constraints:

- **C1.** The voting protocol *must* be compatible with existing Swiss RPV procedures.
- **C2.** The system *must* allow EV procedures, as in the Swiss RPV system.
- **C3.** RPV procedures *must* be resilient to computer infrastructure outages.
- **C4.** The voting processes *must* be compatible with on-site ballot casting.
- **C5.** The system *should* use a decentralized to store evidence.

Fulfilling the listed requirements and their corresponding use cases (*cf.*, Figure 6.9) requires the definition of a protocol, which is presented in the next section.

6.2.2 Voting Protocol

The Helverify protocol follows the phases defined in Section 5.2 [180], as these phases represent the current procedures of the Swiss RPV. Furthermore, the mechanisms applied in these phases follow the suggestions of STROBE-Voting [22]. Figure 6.10 depicts the Helverify protocol phases.

Figure 6.11 shows a high-level overview of the process of running an election with Helverify. In particular, it shows the required steps from the VA's, as well as the voter's perspective, including their interactions. The following sections delve into the steps of this process in more detail.

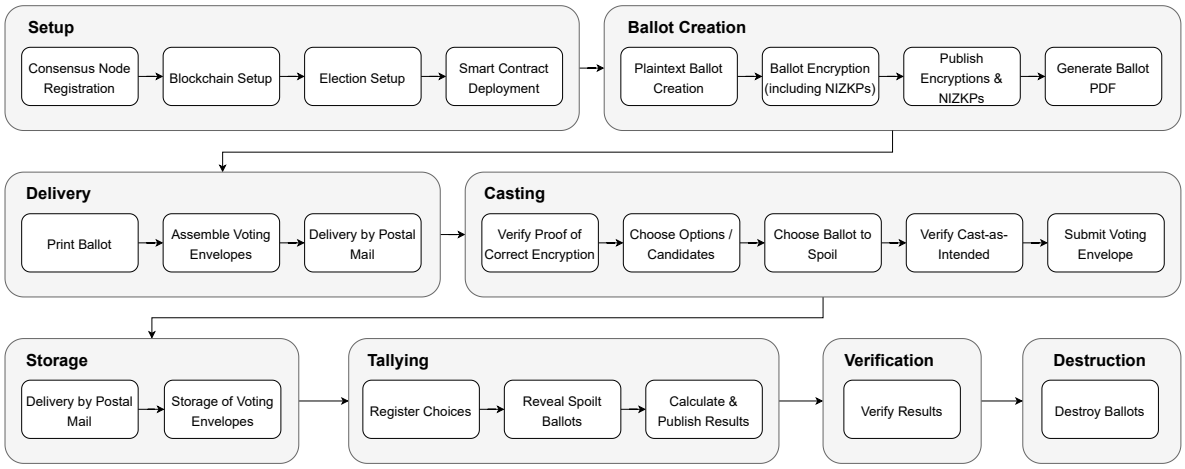


Figure 6.10: HELVERIFY Protocol Phases

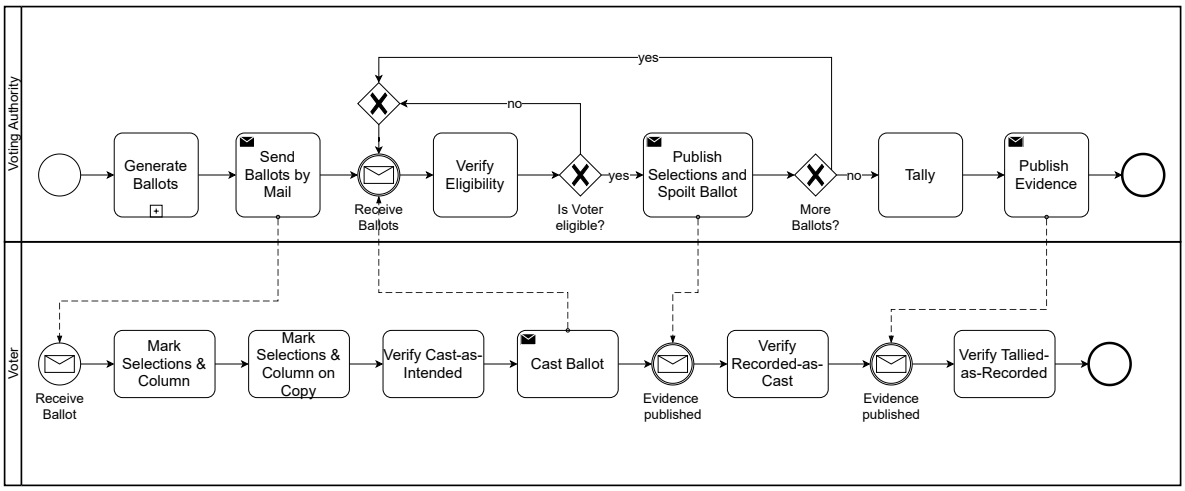


Figure 6.11: Simplified Process Flow of Running an Election in HELVERIFY

SETUP PHASE (1/8)

At the beginning of the setup phase (*cf.*, Figure 6.12), the VA sets up a PoA BC in cooperation with multiple municipalities. This design choice is inspired by Provotum [179], which follows a similar approach. The municipalities are responsible for running Consensus Nodes participating in the public permissioned PoA BC. Once the municipalities have registered their Consensus Nodes, the VA generates a new BC configuration, which it distributes to the Consensus Nodes. After successfully bootstrapping the BC, the VA creates a new election by specifying the parameters, such as candidates/options, title, and encryption.

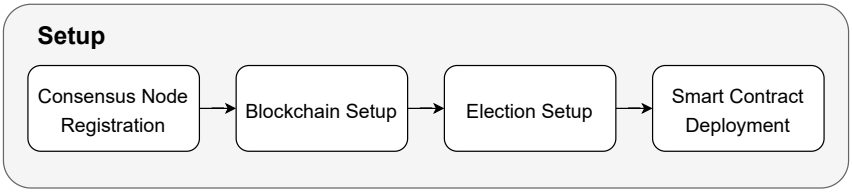


Figure 6.12: HELVERIFY Protocol Setup Phase

Furthermore, the VA requests each Consensus Node to create a public-private key pair ( ) and then collects the public keys ( ) for all Consensus Nodes. Next, the VA uses these public keys to create a composite (election) public key ($_e$). For this Distributed Key Generation (DKG) process, each Consensus Node has to provide proof that it owns the private key to the submitted , which is done by generating a Schnorr Proof [269]. Finally, the Smart Contract (SC) for the election is deployed on the BC. At this point, the BC is ready to persist election evidence.

BALLOT CREATION PHASE (2/8)

Next, the VA (or a delegated TTP) creates the ballots (*cf.*, Figure 6.13). The ballot design corresponds to the one proposed in the STROBE-Voting scheme [22]: Each paper ballot consists of a pair of 'virtual' ballots. A virtual ballot contains a vector of zeros (0) and ones (1) for each candidate or option, where the position inside the vector designates an option or candidate. If the value of a vector element is equal to one (1), it indicates that this vector represents the candidate at the element's position. A valid ballot must only contain one occurrence of one per row and per column, respectively. Each value inside this vector is encrypted with the election public key $_e$ using the exponential ElGamal cryptosystem. To prove the correctness of the created ballot, the VA must generate the following (*i.e.*, Chaum-Pedersen Proofs (CPP)) for each ballot and publish them to the [22]:

- CPP [50] proving that the sum of the encrypted values in each *row* equals one.
- CPP [50] proving that the sum of the encrypted values in each *column* equals one.
- Disjunctive CPP [64]) that each encrypted value is either an encryption of zero or one.

The virtual ballots also contain shortcodes and a ballot code, which are created as follows [22]: The VA produces a hash digest of each vector of encryption and takes the first two characters of the resulting hexadecimal representation. These two characters serve as shortcodes, which are printed next to the corresponding candidate or option on the ballot. The ballot code is the hash digest of all encrypted vectors of both virtual ballots, and it is also printed onto the ballot. Finally, the VA

publishes the encrypted ballots and the corresponding evidence on the [22]. The final step of the setup phase consists of generating PDF files of the ballots that are suitable for physical printing.

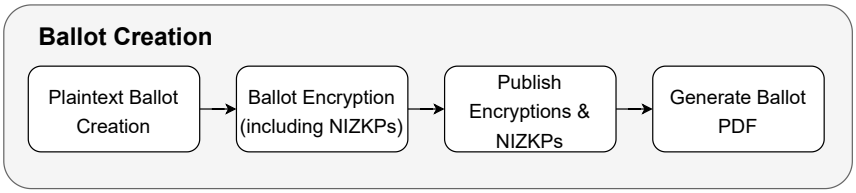


Figure 6.13: HELVERIFY Protocol Ballot Creation Phase

DELIVERY (3/8)

The first step in the delivery phase (*cf.*, Figure 6.14) is to print the ballot PDFs on paper. Following the Swiss RPV process [180], the resulting PBs, PBE, and VSC are placed inside the VE. The Swiss Post then delivers the assembled ballots to the voters. Thus, the delivery phase is congruent with Section 5.2.

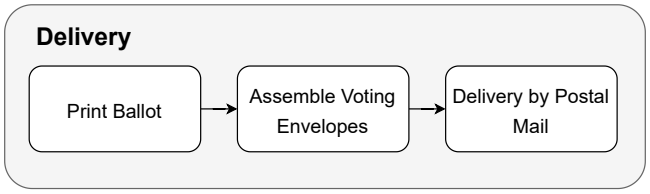


Figure 6.14: HELVERIFY Protocol Delivery Phase

CASTING (4/8)

The Casting Phase encompasses the ballot casting process (*cf.*, Figure 6.15). As soon as the voter receives the VE, she can verify that the ballots have been correctly encrypted by verifying the proofs from the [22]. If the encryption is correct, the voter chooses a candidate/option by marking the respective checkbox on the PB. The remainder of this phase follows the current Swiss RPV procedures (*cf.*, Section 5.2), with minor additions from [22] for verifiability: If the voter wants to perform RaC and TaR verification at the end of the election, she crosses the checkbox of one of the two short code columns, indicating that her selection from this set of shortcodes must be published to the (*cf.*, single ballot variant in [22]). The voter memorizes the shortcode(s) of the selected option(s) and the ballot code for later verification. Furthermore, the voter notes the shortcodes of the second

(unselected) column and their corresponding options/candidates. At this point, the voter verifies that she has selected the intended option (CaI). Next, the voter signs the VSC and puts the PB into the PBE and seals it. Finally, the voter puts the signed VSC and the sealed PBE into the VE, seals it, and sends the VE to the municipality by postal mail.

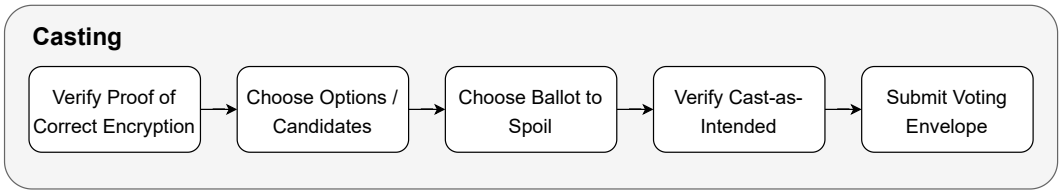


Figure 6.15: Helverify Protocol Casting Phase

Storage 5/8

The storage phase (*cf.*, Figure 6.16) remains unaltered from the original process described in [180]. In other words, the municipalities store the VEs sent by the voters until the tallying process begins.

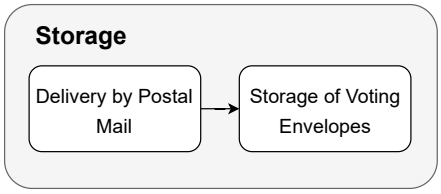


Figure 6.16: Helverify Protocol Storage Phase

Tallying 6/8

The first step of the tallying phase (*cf.*, Figure 6.17) is to verify the voter's eligibility by checking the signature on the VSC [180]. Next, the VA publishes two pieces of evidence on the [22]:

- The shortcodes from the selected short code column representing the voter's choice.
- For the other set of shortcodes (*i.e.*, which the voter did not select), the corresponding decrypted virtual ballot containing the options (in plaintext with its corresponding shortcodes).

The published shortcode(s) enable the voter to verify that her vote has been RaC and the decrypted virtual ballot allows for verifying that the shortcodes correspond to the correct options. Note that a malicious VA will be caught producing incorrect shortcodes with the probability of $\frac{1}{2}$ for each ballot,

as it does not know in advance which virtual ballot the voter chose. Thus, for n validating voters, the probability of detecting a cheating VA is $\frac{1}{2^n}$ [22]. To determine the final results of the election, the VA performs a homomorphic addition of all selected options, which yields a ciphertext containing the tallies for each candidate/option. To retrieve the result in plaintext, Helverify follows the approach of Provotum [179] in applying Cooperative Decryption (CD): The VA sends the ciphertext to the Consensus Nodes, which produce a decrypted share by decrypting the ciphertext with their private keys. The Consensus Nodes also generate a proof of correct decryption (Chaum-Pedersen Proof [50]). The VA verifies the proofs and then combines the shares, which yields the final result in plaintext. Finally, the VA publishes the results on the [22].

Verification (7/8)

In the verification phase (*cf.*, Figure 6.17), the published results and the corresponding evidence [22] allows voters and observers to perform RaC and TaR verification.

Destruction (8/8)

The destruction phase (*cf.*, Figure 6.17) involves the physical destruction of the PBs and VSCs [180] and the tear-down of the PoA BC.

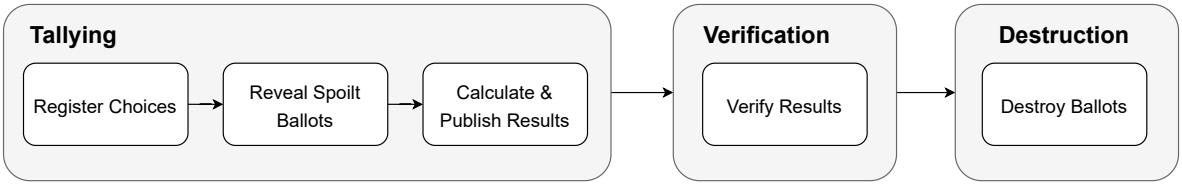


Figure 6.17: Helverify Protocol Tallying, Verification, and Destruction Phases

6.2.3 Architecture of Helverify

This section provides an overview of Helverify's architecture, consisting of the components depicted in Figure 6.18.

Voting Authority

The VA component consists of a frontend with a corresponding backend. The frontend allows election officials to set up the BC infrastructure and elections. Officials can generate ballots and export

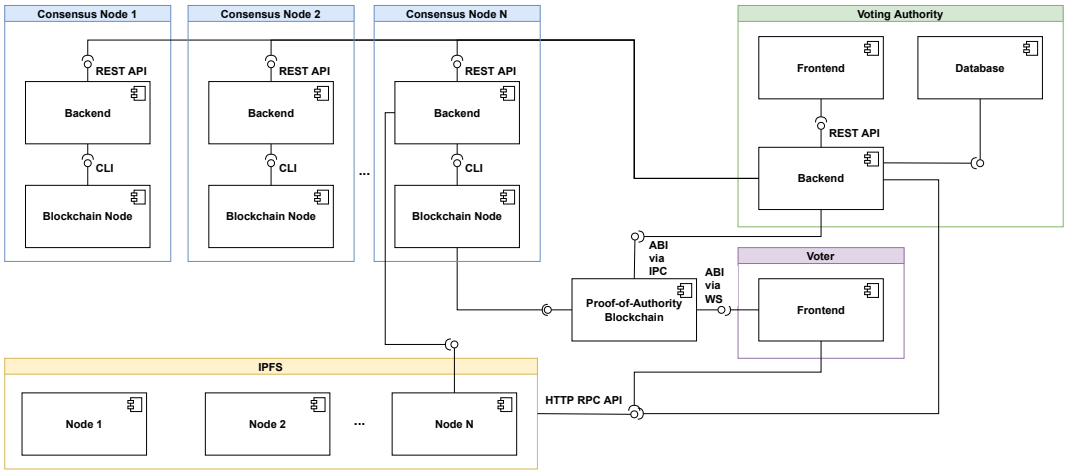


Figure 6.18: Architecture of HELVERIFY

them as PDF files. In the tallying phase, the VA component allows for registering ballot selections while publishing evidence of the process. Finally, election officials can initiate the tallying process. For this purpose, the ballot selections are homomorphically added and then cooperatively decrypted by the Consensus Nodes. Lastly, the results are persisted on to the election SC.

CONSENSUS NODE

The Consensus Nodes serve two purposes: Operating the PoA BC, which hosts the election SC, and participating in the DKG and CD processes for ballot encryption and decryption. Each Consensus Node contains a 'BC Node'. The BC Nodes of all Consensus Nodes are responsible for operating the BC network. The Consensus Node Backend orchestrates both the BC node and the cryptographic operations by providing an API, which the VA Backend consumes.

INTERPLANETARY FILE SYSTEM

Depending on the key length of the ElGamal cryptosystem chosen for an election, ciphertext representations of a ballot can exceed the storage capacities of SCs. Thus, HELVERIFY uses a private Interplanetary File System (IPFS) [155] swarm to store encryptions and proofs for the ballots, improving the scalability of the solution. The VA Backend stores and retrieves encryptions and evidence (*i.e.*, proofs). Finally, the Voter Frontend uses the IPFS swarm to retrieve evidence for verification.

Voter

The Voter Frontend allows the voter to perform multiple forms of verification. Firstly, the voter can verify that the ballot she received has been properly encrypted by checking the proof of correct encryption for each of the encrypted values on the ballot. After the VA has registered the selected choices from the ballot, the voter can verify that the VA has registered the the ballot choices correctly. In addition, the spoilt ballot verification allows the voter to be assured that the ballot encryption has been performed correctly. Thus, the Voter Frontend allows the voter to perform the verification steps described in [22].

Proof-of-Authority Blockchain

Helverify uses an Ethereum BC for hosting its . Due to the scalability and privacy requirements, Helverify uses a BC with the PoA consensus algorithm. As mentioned earlier, the Consensus Nodes operate the BC. Helverify's BC is a platform for hosting the election SC, which provides the functionality for a particular election. The election SC stores references to the storage locations of ballots on IPFS, the selections of a particular ballot, and the results of an election.

6.2.4 Implementation

Helverify was implemented in C# with ASP.NET 6 [258] for the backend applications, providing REST APIs (including cryptographic primitives), TypeScript [220] with React [219] for the frontend applications, Solidity [284] for the SC, and Docker [88] for containerizing the individual components. The cryptographic library is implemented in C#, based on the Bouncy Castle library [201], including both the standard and exponential variants of the ElGamal cryptosystem.

The Helverify cryptography library also contains the implementations of four types of NIZKPs: Namely, proof of private key ownership, proof of correct decryption, proof of containing zero or one, and proof of containing one. Each proof consists of a proof generation (*e.g.*, See Listing 4 in Appendix C) and a verification method (*e.g.*, Listing 5 in Appendix C).

Consensus and Blockchain Nodes

The Consensus Nodes participate in the PoA network, forming the . Each Consensus Node exposes a REST API for configuration, key pair management, and cooperative decryption, since Consensus Nodes serve as a trusted entity for DKG and Cooperative Decryption (CD) in an election. The Blockchain Node is a collection of scripts that control the Go Ethereum (`geth`) client on the

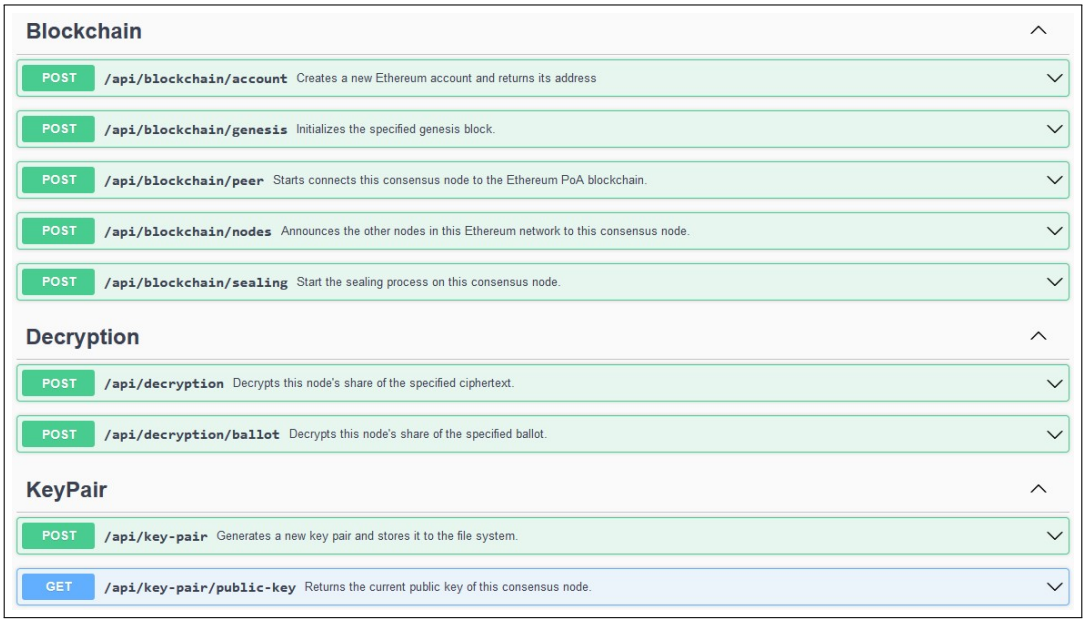


Figure 6.19: Consensus Node Backend REST API

Consensus Node. Therefore, the backend of the Consensus Node is controlled via the REST API interacting with `geth` through invoking scripts with specific parameters *i.e.*, the structure of the API largely follows the structure of the scripts(*cf.*, Figure 6.19). Each Consensus Node holds an ElGamal key pair per election because an election's public key consists of the combination of all the Consensus Nodes' public keys.

Voting Authority

The Voting Authority (VA) application consists of a backend and frontend component. The VA backend exposes a REST API to the VA frontend. The VA backend covers BC setup, election management, ballot recording, and evidence publishing. While the essential endpoints are explained in the following sections, a complete overview of the VA backend REST API is shown in Figure C.3 in Appendix C.

The BC must be set up once and is usable for multiple elections. The VA backend provides a REST API endpoint (*i.e.*, `/api/blockchain`), which allows for the creation of a new BC network on the specified Consensus Nodes. The setup process (*cf.*, Figure C.1 in Appendix C) consists of multiple steps initiated by the Consensus Node backend. After these steps, the BC is ready for the deployment of SCs. Thus, the next step is to set up an election.

Election Setup During *Election Setup*, a three-step process involves first (*cf.*, Figure C.2 in Appendix C), where *(i)* first, election data (*i.e.*, question, candidates/options, cryptographic parame-

ters) is persisted to the VA backend via a REST API call (`/api/elections`). This data is then stored in a MongoDB database. Second *(ii)* the `/api/elections/{id}/public-key` endpoint is called, which leads the VA backend to call every registered Consensus Node to generate an ElGamal key pair for this election. After receiving the public keys of all Consensus Nodes, the VA backend combines them into one composite election public key (*i.e.*, Distributed Key Generation (DKG)), which it stores on the MongoDB database. Lastly, *(iii)* the `/api/elections/{id}/contract` endpoint is called, and the VA backend deploys the election SC onto the PoA BC. This step also includes initializing the SC by registering the election information, such as candidates/options, ElGamal parameters, and all public keys, both of the election and all Consensus Nodes.

Ballot Generation In *Ballot Generation*, the first step is to generate a ballot template from the election parameters, which is required to generate the encrypted ballots (also referred to as virtual ballots). The VA backend persists these virtual (encrypted) ballots to IPFS, only storing the Content Identifiers (CID) [154] on the MongoDB backend, while the plaintext ballots are stored on the MongoDB for printing.

Data Model and Ballot Formats The relationships between different ballot representations is depicted in Figure 6.20. Each election has a dedicated `BallotTemplate`. The `BallotTemplate` contains a reference to the `Election` to which it belongs.

Furthermore, it contains a list of plaintext options representing the options/candidates for this particular election (*i.e.*, `PlainTextOption`). A `PlainTextOption` represents one candidate/option, consisting of the candidate's name, the shortcode (after encryption), and a vector of zeros and a one, where the one represents the current candidate. The `BallotTemplate` can encrypt the plaintext options to yield a `VirtualBallot`. The `VirtualBallot` consists of plain- and ciphertext representations of all candidates/options.

The `EncryptedOptions` consist of the shortcode, the hash of the encryptions, and the encryptions the number vector (`EncryptedOptionValue`, consisting of the ElGamal ciphertext and a proof that the encryption contains either the value zero or one). The `VirtualBallot` also contains the proofs for guaranteeing that each candidate is only represented once and that each vector of numbers only represents one candidate. Furthermore, the `VirtualBallot` is only usable if all generated shortcodes are unique [22].

When storing a virtual ballot to IPFS, the plaintext values are not included, as these values would compromise ballot privacy. The construction of a `PaperBallot` takes two `VirtualBallots`. Note that the `PaperBallot` contains all options in plaintext, along with the shortcodes, ballot codes,

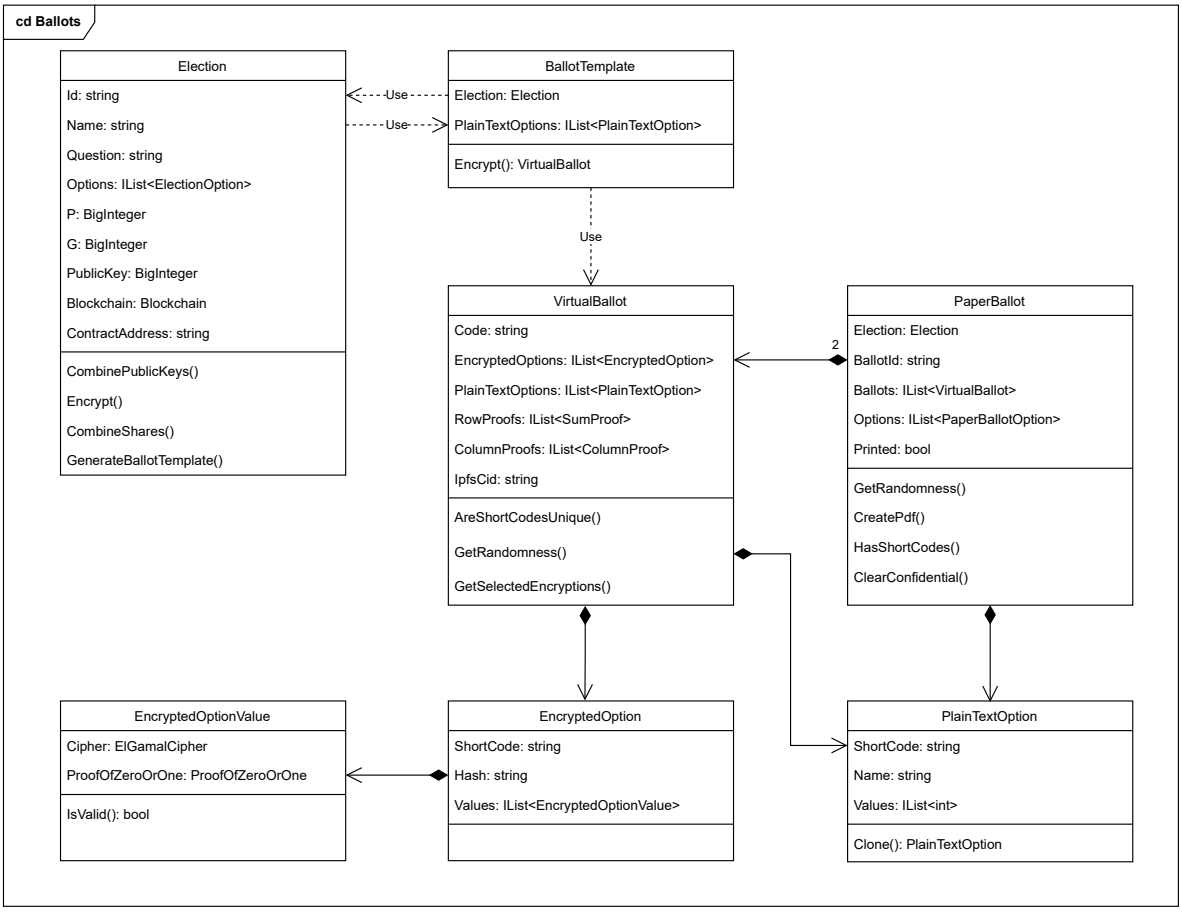


Figure 6.20: Class diagram showing the structure and composition of ballots in plaintext and encrypted states.

and ballot ID, as this information is needed for printing ballots later. Each option/candidate has an accompanying shortcode. A shortcode is derived as follows (*cf.*, Listing 6, lines 19-23): First, the SHA-256 hash of all encrypted values of the option is calculated and then reduced to the first two characters. In case of duplicate shortcodes in a `VirtualBallot`, the ballot is discarded and a new `VirtualBallot` is generated.

Ballot Recording The `/api/elections/{electionId}/ballots/{ballotId}/evidence` endpoint serves the purpose of registering the voter's choice as marked on the printed paper ballot. Additionally, the VA backend publishes evidence in the form of the spoilt ballot marked by the voter. The process of recording ballot choices (*cf.*, Figure C.5) works as follows:

1. First, the respective paper ballot is loaded from the database.
2. The VA backend retrieves the IPFS references (CID) of the ballot from the contract.

3. Next, the voter's selections (*i.e.*, the selected shortcodes of the chosen code column) are persisted on the election SC.
4. The other column (*i.e.*, the column that has not been selected by the voter) indicates the virtual ballot to be spoilt. Spoiling a ballot involves decrypting the values before storing the ballot as plaintext on the PBB.
5. The Consensus Nodes cooperatively decrypt the virtual ballot to be spoilt. For this purpose, each Consensus Node retrieves the encryptions from IPFS by using the IPFS CID of the ballot to be spoilt and then produces a decrypted share, which the VA backend collects.
6. The VA backend combines the decrypted shares to restore the original plaintext of the virtual ballot.
7. The restored virtual ballot is published on IPFS.
8. The IPFS CID of the spoilt ballot is published on the election SC.

Tallying and Publishing Evidence The last feature of the VA backend is tallying and publishing the results of an election (*cf.*, Figure 6.21), including evidence that the results are correct. A call to the `/api/elections/{id}/tally` endpoint starts the following process:

1. The election data is loaded from the database.
2. The VA backend retrieves the number of cast ballots from the election SC.
3. The VA backend loads each ballot ID from the election SC in this step. With the ballot ID, the VA backend can retrieve the ballot references from the election SC, including the selected options/candidates from the ballot.
4. Using the ballot references, the VA backend loads the selected options of each ballot from IPFS in encrypted form.
5. The VA backend adds all encrypted selections homomorphically, yielding one ciphertext per candidate containing the tally for said candidate.
6. Each candidate's ciphertext is cooperatively decrypted by all Consensus Nodes, yielding the tally for each candidate after the VA backend has combined the decrypted shares. The decrypted shares also include the proofs of correct decryption.
7. The VA backend publishes the results and the corresponding proofs on IPFS.
8. The last step consists of the VA backend storing the IPFS reference (*i.e.*, CID) and the final results on the election SC. Thus, the results are now available for verification.

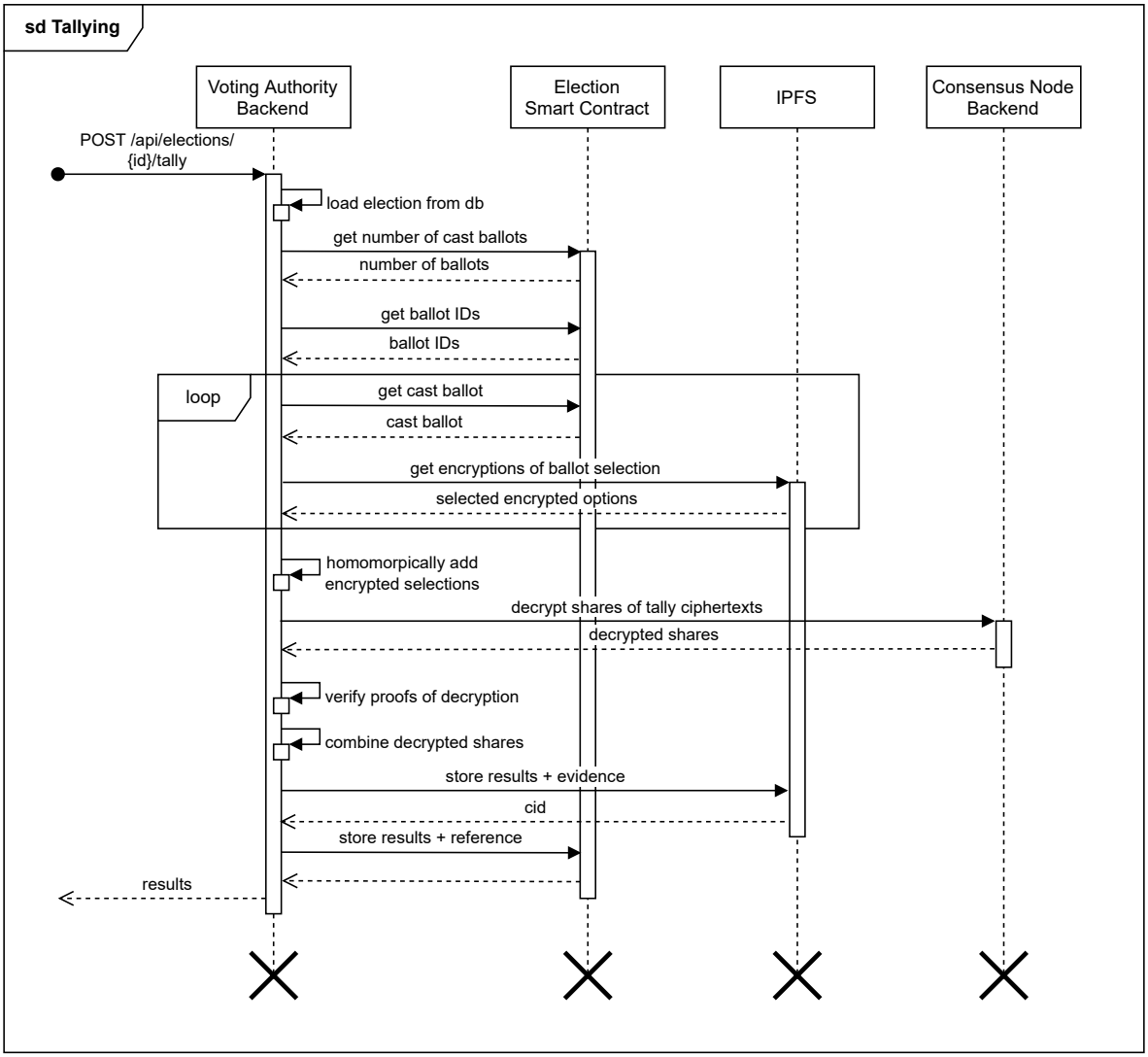


Figure 6.21: Sequence diagram showing the tallying and evidence publishing process of an election.

VA Frontend

The VA frontend allows the election organizers to set up elections and handle all life-cycle events of an election. The frontend is implemented in React [219] using TypeScript [220]. Figure 6.22 depicts the VA frontend, particularly the detailed view of a specific election. The VA frontend can handle all of the relevant use cases for election management. Election officials can set up new elections via a step-by-step process, guiding the user through every aspect of an election.

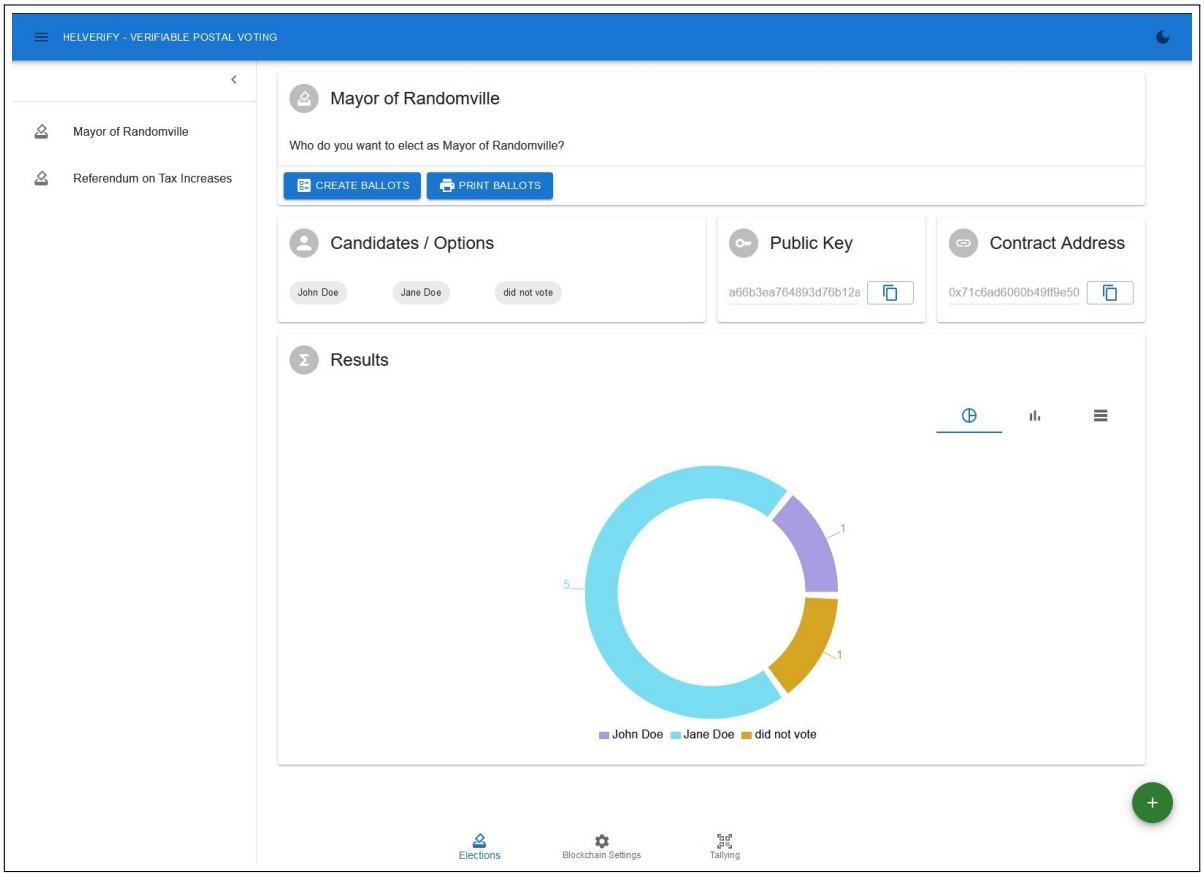


Figure 6.22: Voting Authority UI

Ballot Generation and Printing Ballot generation and printing is handled through dedicated VA frontend modal dialogues. Figure 6.23 shows an example of a paper ballot. A paper ballot contains the election name, the questions, and the options/candidates. The ballot also contains a QR code in the top right corner, which provides a user-friendly way to read the ballot's data. The unique ballot ID is printed below the options, identifying each ballot and serves as an aid in distinguishing individual ballots.

A voter is required to select one of the two columns. The shortcodes (*e.g.*, c5) will be published after the ballots have been recorded by the election offices. The other column (*e.g.*, the one starting with 8c) is spoilt, meaning that the virtual ballot will be decrypted and published together with the respective shortcodes, allowing the public to verify that the ballot's encryption was correct [22]. The printed ballots are in PDF format, consisting of two pages: The first page contains the actual ballot. The second page, however, contains a watermarked copy, which the voter can keep and use to mem-

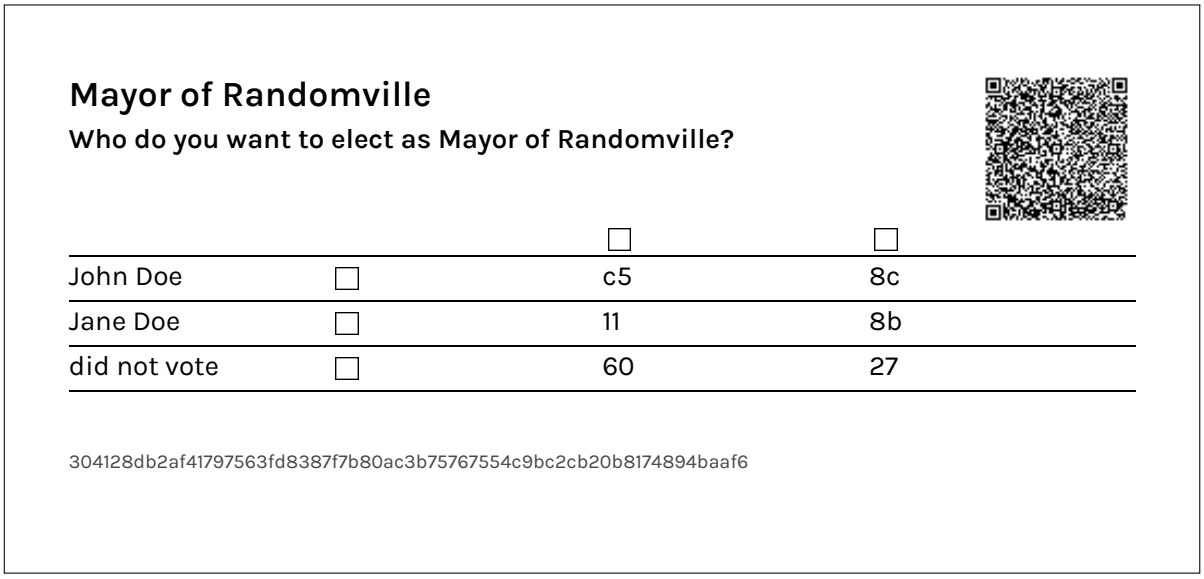


Figure 6.23: Printed Ballot

orize her choices for verification later on. The system provides all ballots generated in a archived file, which can be handed to a service partner for printing.

Ballot Recording Once the voters have selected their choices and sent the ballots back to the VA, the election officials register the choices marked on the ballot with Helverify. For this purpose, an election official scans the QR code on the ballot and enters the user's selection into the ballot recording form. This form is a digital version of the paper ballot, allowing the officials to enter the voter's selections and the selected column. This process could be considered inefficient and error-prone in real-world scenarios, as it relies on manually entering the voter's choices. However, an automated OCR approach could be implemented in such scenarios for more efficient ballot recording (*e.g.*, similar to so-called E-Counting methods).

Once the VA has finished registering all ballot selections, the officials can use the VA frontend to calculate the final results and publish them with the corresponding evidence. After publishing the results, the VA and voter frontends visualize the election outcomes by providing multiple visualizations (*i.e.*, pie chart, bar charts, and a table, *cf.*, Figure 6.22).

Voter Frontend

The voter frontend allows the voter to verify the election proceedings, with different means of verification available in each phase of an election. Furthermore, the voter frontend interacts directly with the election SC and the IPFS nodes to access evidence and ballots.

When receiving the paper ballot by postal mail, the voter can verify that the ballot is well-formed and properly encrypted. First, the voter scans the QR code on the ballot (or the ballot copy), which loads the required data from the election SC (*i.e.*, the ballot references) and the IPFS network (*i.e.*, the ballot data). The voter frontend offers the possibility to to perform this verification(*cf.*, Fig 6.24).

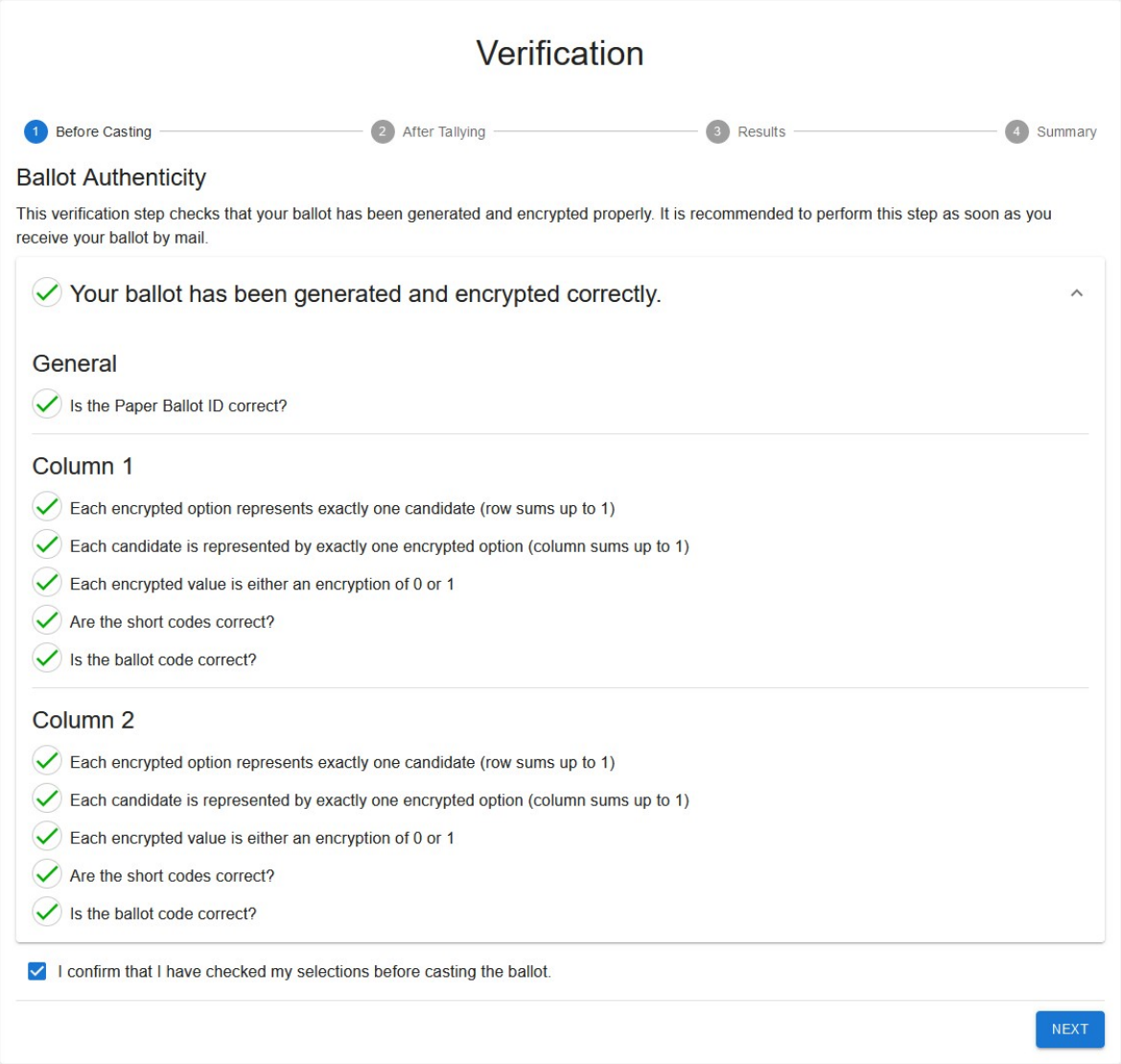


Figure 6.24: Ballot Authenticity Verification

Figure 6.24 details all verification results, showing details on-demand, hiding unnecessary complexity by default to avoid confusion. The verification includes the following checks (according to the mechanisms defined in [22]):

- The ballot ID is correct, meaning it results from hashing the ballot codes of both virtual ballots.
- For each virtual ballot:
    1. Each set of encryptions (*i.e.*, vector of a candidate/option) represents precisely one candidate/option.
    2. Each option/candidate is only represented once.

3. Each ciphertext contains either the value 0 or 1.
4. The shortcodes are correct (*i.e.*, the two first characters of the SHA-256 hash of the corresponding ciphertexts).
5. The ballot code is correct (*i.e.*, it is the SHA-256 hash of all encryptions).

The verification algorithms of the proofs mentioned above have been implemented in TypeScript. Listing 7 in Appendix C shows the TypeScript implementation of the proof that a ciphertext either contains the value 0 or 1. The implementations of the other proof verification algorithms follow the same principle.

Ballot Choice and Spoilt Ballot

Another verification procedure consists of checks that the VA has registered the voter's choices correctly and that the spoilt ballot's shortcodes match the shortcodes on the ballot copy [22].

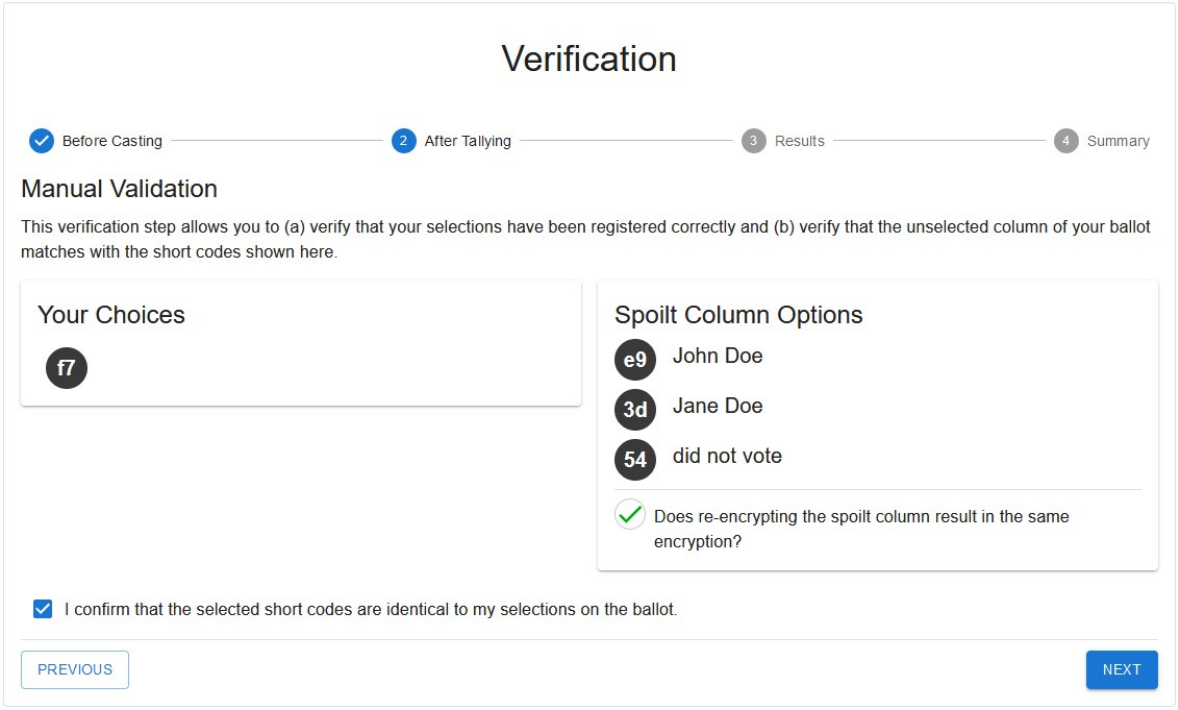


Figure 6.25: Choice and Spoilt Ballot Verification

Figure 6.25 shows this verification step on the voter frontend. First, the voter checks that the shortcodes displayed in the voter frontend match the selected options on the selected column of her ballot. If they match, the voter can be sure that her choices are registered correctly. As suggested by

STROBE-Voting [22], the spoilt ballot allows the voter to expose a cheating VA with a probability of $\frac{1}{2}$. Additionally, the voter frontend re-encrypts the spoilt ballot to check whether the resulting ciphertext matches the original ciphertext stored on IPFS.

Display of Results and Summary After verifying the ballot choices, the voter can inspect the election results. This step includes the verification of the proofs that the results have been decrypted correctly. Finally, a summary is presented to the voter (*cf.*, Figure 6.26), which shows the properties achieved during the verification. The summary marks the end of the verification process.

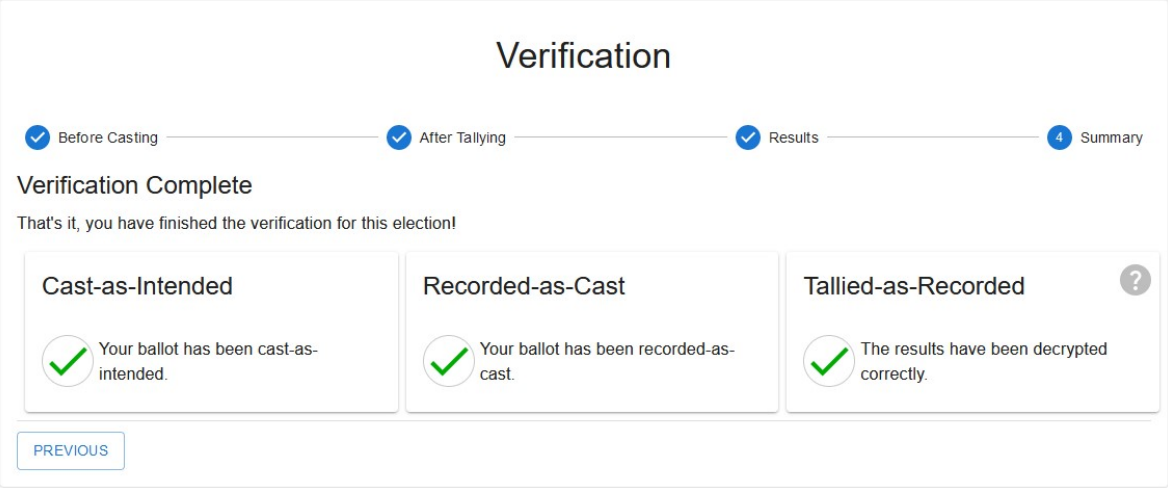


Figure 6.26: Verification Summary

6.2.5 Performance and Scalability Evaluation

This section focuses on the evaluation of the performance and scalability of Helverify concerning ballot creation and tallying, which depends heavily on cryptographic operations. Specifically, the focus is on the dependency of specific variables (*i.e.*, number of options, key length) on the runtime duration for specific use cases for varying numbers of ballots (*i.e.*, 1'000, 10'000, 100'000, and 1'000'000). In particular, the scenarios consist of *(i)* creating ballots and *(ii)* calculating the final results. All of the measurements are based on Helverify with a block time of eight seconds and a gas limit of 180'000'000, which proved to be an optimal trade-off of reliability and performance. The performance figures in this section have been measured on an AMD Ryzen 5950X (16 cores / 32 threads) with 64 GB of system memory available, running Ubuntu 22.04 LTS with Docker [88]. Further, a private IPFS swarm was used for all measurements, with the data being stored on a single

IPFS node. The reason for this setup lies in the reproducibility of the results, as it allows for spotting differences in storage duration caused by the variables in the measurements.

Ballot Creation

The two main factors of importance for creating ballots are the cryptographic key length and the number of options/candidates available for an election. Figure C.6 shows the ballot creation duration in relation to the number of ballots created with 2048-bit and 4096-bit keys, respectively. Within each series of measurements (*cf.*, Table 6.3) of the same key length, the relation between the number of ballots and duration is roughly linear. However, a slight performance decrease is noticeable: For 1'000 ballots at 2048-bit key length, each ballot is created in 0.0172 seconds, while at 1'000'000 ballots, the time increases to 0.02868 seconds per ballot. The same pattern can be observed for 4096-bit key length (*i.e.*, 0.045 seconds per ballot at 1'000 ballots, 0.05682 seconds per ballot at 1'000'000 ballots). This effect could partially be due to performance advantages for short workloads, as the CPU can clock at higher frequencies during short workloads. When comparing ballot creation duration with 2048-bit and 4096-bit key lengths, it can be observed that the ratio between these durations decreases with an increasing number of ballots (*i.e.*, 2.61 at 1'000, 1.98 at 1'000'000). This behavior could stem from the block time of the private chain, which introduces variance in the results, especially noticeable for low numbers of ballots *i.e.*, the block time has more impact on short timespans, which is the case for 1'000 and 10'000 ballots. Thus, increasing the number of ballots created reduces this effect.

Table 6.3: Duration measurements in minutes for creating ballots

Scenario	1'000	10'000	100'000	1'000'000
2048-bit 3 options	0.29	3.27	40.50	478.00
4096-bit 3 options	0.75	7.85	93.10	947.00
2048-bit 6 options	0.85	8.92	106.58	1'117.00

Figure 6.28 depicts the durations for creating ballots depending on the number of options/candidates specified for the election. Again, the durations for creating 1'000, 10'000, 100'000, and 1'000'000 ballots were measured (*cf.*, Table 6.3). When changing the number of options from three to six, the time for creating 1'000'000 ballots increases by 133.68%. As a major part of ballot creation consists of generating NIZKPs, the number of proofs to be created impacts the duration of ballot creation significantly: In Helverify, the number of proofs required for n options can be described by the polynomial $n^2 \quad 2n$ (*cf.*, Figure 6.27). Consequently, for three options ($n \quad 3$), one ballot requires

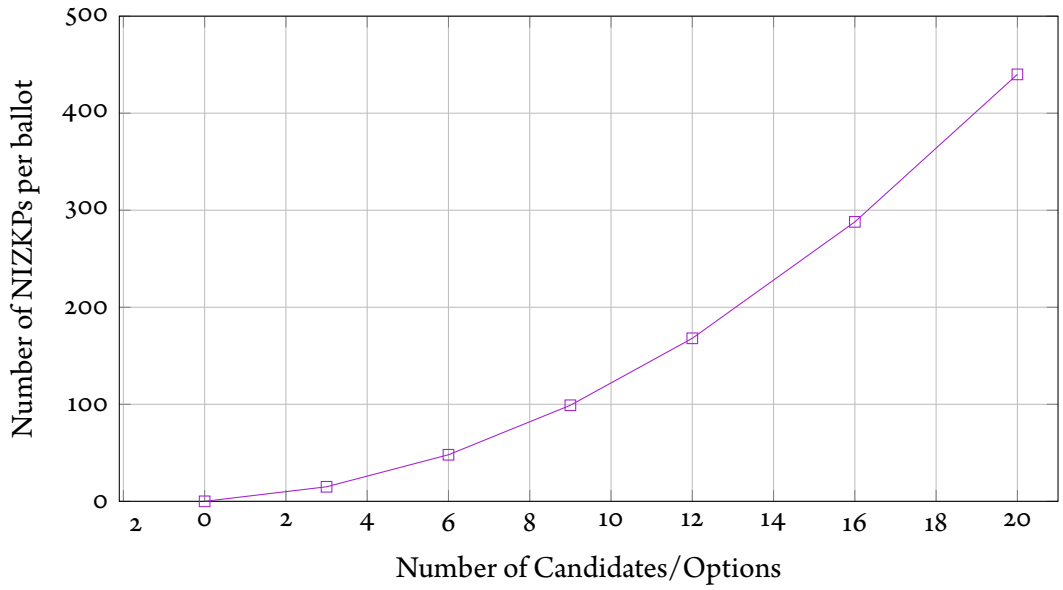


Figure 6.27: Number of NIZKPs required for number of options/candidates

15 proofs, while for six options (n 6), each ballot requires 48 proofs. Other factors, such as BC transaction processing, remain identical, as in this case, the key length did not change.

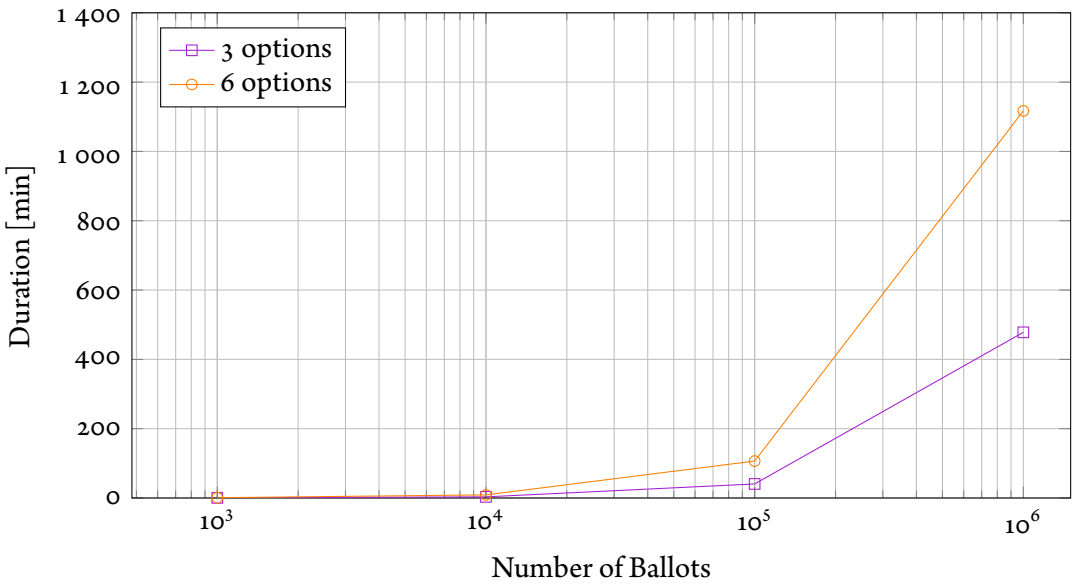


Figure 6.28: Ballot creation duration for different number of options (2048-bit key length)

Result Calculation

In order to perform a realistic performance and scalability analysis for the result calculation, Helverifymust be in a system state where registered votes are present. For that reason, a dedicated VA backend endpoint for evaluation was implemented, which allows selecting random choices for each ballot. As in the previous analysis, the measurements consist of execution rounds with 1'000, 10'000, 100'000, and 1'000'000 ballots, respectively (*cf.*, Table 6.4). Furthermore, the variables remained the same (*i.e.*, key length and the number of options/candidates).

Table 6.4: Duration measurements in minutes for tallying ballots

Scenario	1'000	10'000	100'000	1'000'000
2048-bit 3 options	0.08	0.59	6.05	61.30
4096-bit 3 options	0.34	2.05	20.60	215.00
2048-bit 6 options	0.15	1.10	11.28	113.07

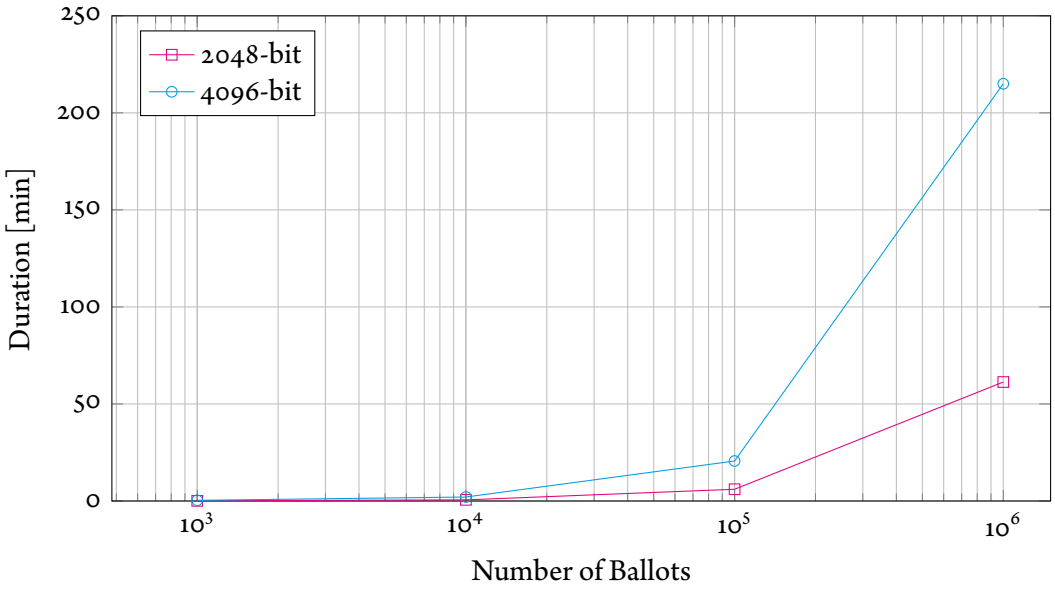


Figure 6.29: Tallying duration for different key lengths (3 options)

Figure 6.29 shows the results of tallying specific numbers of ballots with varying key lengths, while keeping the three options/candidates constant. The results show that using longer keys comes with a performance penalty: This is mainly due to the computational effort of performing homomorphic addition, which takes between 78.81% (2048-bit key, 10'000 ballots) and 81.31% (4096-bit key, 10'000 ballots) of the total result calculation duration. In this case, doubling the key length (2048-bit to

4096-bit) increases the duration from 61.3 to 215 minutes (+250.7%) for tallying 1'000'000 ballots. Furthermore, the data size of a ballot increases with growing key lengths, which contributes to the longer ballot creation durations. On the other hand, other interactions, such as SC and MongoDB calls, remain the same for varying key lengths and are, thus, negligible. Lastly, the time for cooperative decryption can vary, as the decryption is parallelized in a divide-and-conquer manner. This mechanism was implemented to optimize decryption, leveraging the capabilities of multi-core CPUs. Consequently, for some tallies, the decryption can be faster than others, mainly influenced by the partition borders for brute-forcing the exponential ElGamal ciphertext.

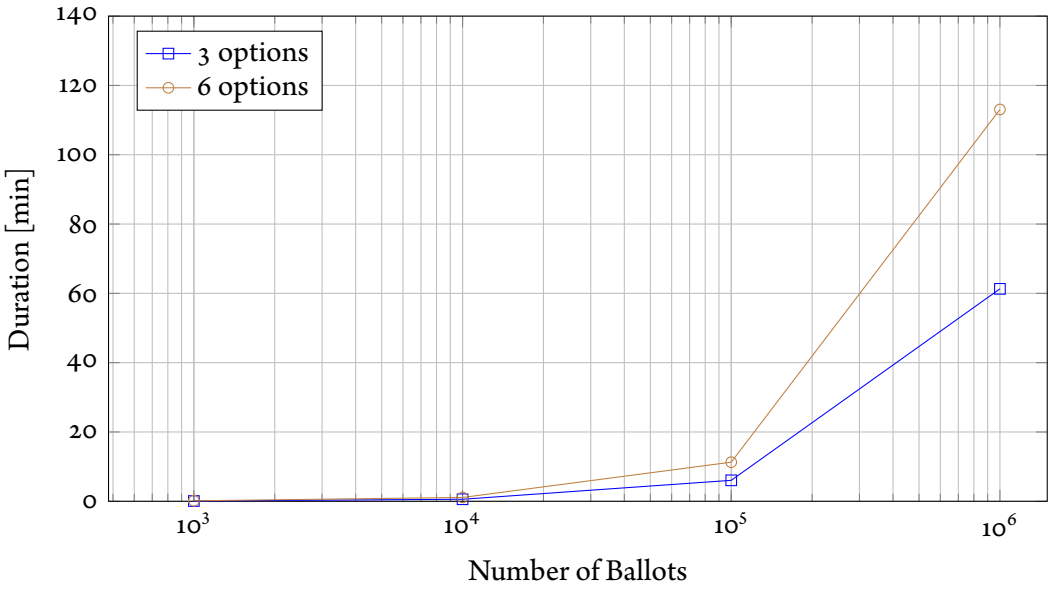


Figure 6.30: Tallying duration for different number of options (2048-bit key length)

Increasing the number of options/candidates (*cf.*, Figure 6.30 and Table 6.4) does not come with the same performance penalty as ballot creation. At 1'000'000 ballots, the time to calculate the final results increases from 61.3 (3 options, 2048-bit key) to 113 minutes (6 options, 2048-bit key), which is less than double the time (+84%). Again, this could be due to the influence of the NIZKP creation during ballot generation, which is absent during the result calculation.

6.2.6 Trust Boundaries and Key Lengths

From a security viewpoint, key lengths and trust assumptions are crucial, since they influence the overall security level of any cryptographic protocol. For that reason, the following section delves into the evaluation of Helverify's key length and trust boundaries.

NIST recommends using key lengths of at least 2048-bit for Diffie-Hellman keys [15]. In this respect, Helverify fulfills the requirements: During the evaluation, key lengths from 2048-bit keys up to 8192-bit were evaluated. The measured performance at 2048-bit is decent, there are decreases in performance at 4096-bit and significant impacts when increasing to 8192-bit key length. From a practical point of view, this effect can be reduced by upgrading to server-grade hardware, especially the CPUs.

Security Evaluation of Trust Boundaries

Regarding trust assumptions, Helverify distributes the trust between multiple entities (*cf.*, trust boundaries in Figure 6.31): The PBB contains all the ballot encryptions, selections, spoilt ballots, results, and election evidence. These artifacts are publicly available, thus, the PBB forms a trust zone.

The VA's components are located inside a confidential trust zone, as it contains the PBs in plaintext. Thus, the VA backend is the only component allowed to write into the PBB. Other components, be it in this or other zones, are only allowed to read from the PBB. The Consensus Node zone is the second confidential zone of Helverify. It is classified as confidential because each Consensus Node stores multiple key pairs: The backends each store a key pair per election and a single key pair for the BC account.

Moreover, the verification zone is another publicly accessible part of Helverify. It contains the Voter Frontend, the verification application for the voters. Finally, the voter zone is classified as confidential, as it involves the voter storing the ballot copy, which she needs to verify the election procedures via the Voter Frontend. The ballot copy is considered sensitive data, as it contains the voter's selections and the spoilt column choice, mirroring the original cast ballot.

Based on Figure 6.31, the following trust assumptions can be formulated, which must hold for the system to be secure:

- **Voter**: The voter must keep the ballot (copy) secret. Otherwise, third parties could learn the voter's selection. However, the ballot copy does not necessarily prove how the voter voted, as she could deceive third parties by selecting different choices on the copy than on the original ballot.

- **Voting Authority**: The PBs must be kept secret. Revealing the PBs would mean revealing the mapping between ballot codes and their plaintext options, indirectly exposing voters' selections. Furthermore, the VA must behave honestly.

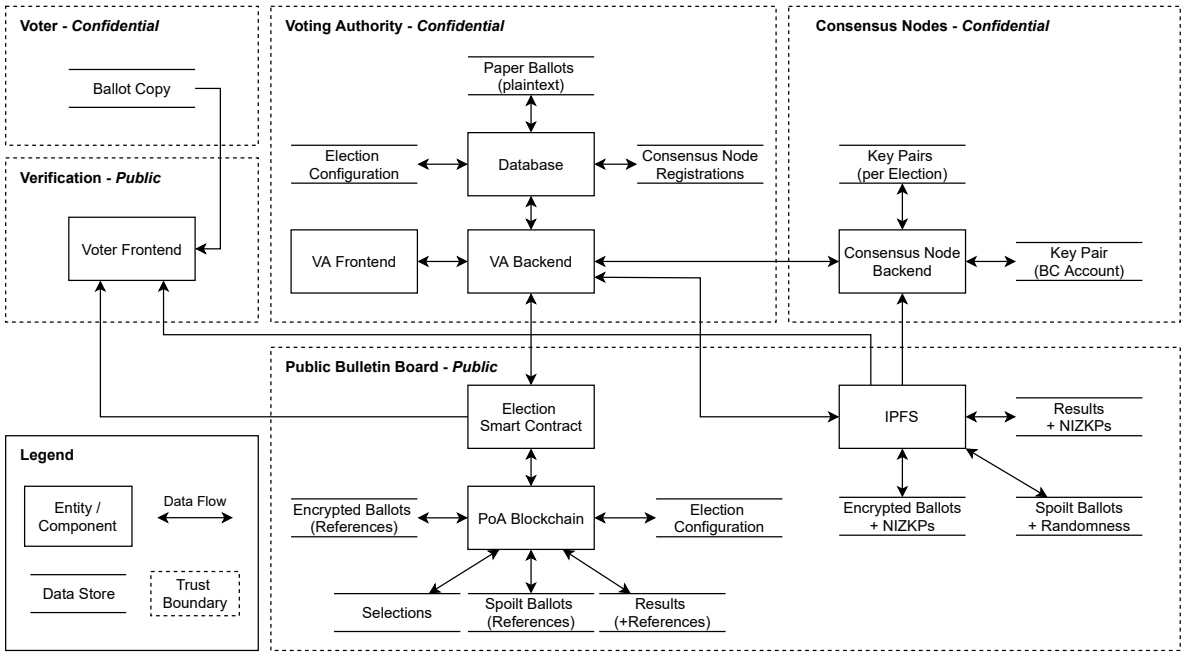


Figure 6.31: HELVERIFY's trust boundaries (notation inspired by [275])

- **Consensus Nodes**: The private keys of all key pairs (*i.e.*, election and BC) must remain secret. Due to applying threshold cryptography, an attacker could decrypt ballots and results only if she learns all Consensus Nodes' private keys for an election. If the BC account private key of a Consensus Node is exposed, an attacker could sign blocks in the name of the Consensus Node. As with the VA, the Consensus Nodes must behave honestly.

- **Public Bulletin Board and Verification**: These zones are public, thus the information is freely available. However, writing to the PBB must be restricted (*i.e.*, only by VA). Additionally, data integrity is guaranteed by the immutability property of the BC and IPFS.

Thus, if these assumptions hold, HELVERIFY can be considered secure. However, this raises the need for mechanisms to detect if these trust assumptions are violated. Hence, the following mechanisms are available for detecting malicious activities and trust violations: The voter can use the evidence on the PBB to verify the VA's compliance with election procedures using the Voter Frontend. Thus, the voter can detect a VA behaving maliciously. Moreover, the voter and VA rely on the Consensus Nodes to act honestly. Hence, using multiple Consensus Nodes applying threshold cryptography (*i.e.*, DKG and CD) prevents a single entity from decrypting ballots, which lowers the trust assumption on each Consensus Node and the VA.

6.2.7 Evaluation of Properties and Requirements

Privacy and verifiability are crucial properties for any voting system. Helverify is based on STROBE-Voting [22] and applied to the Swiss RPV procedures [180]. Therefore, Helverify achieves the same properties as STROBE Voting.

Privacy: Helverify achieves BP by applying HE. Applying HE allows the VA and the Consensus Nodes to calculate the final results without decrypting individual ballots, *except* for the spoilt ballots, which are not associated with selected options/candidates. Furthermore, Helverify fulfills RF, assuming that the voter does not deliberately reveal her choices. Even the voter's ballot copy does not necessarily prove to a coercer that the voter voted in a specific way, as the voter could also create another copy to deceive a coercer. According to [66], STROBE-Voting [22] fulfills RF, which is also the case for Helverify.

Verifiability: Helverify achieves full E2E-V by using the following mechanisms: First, the voter must ensure that the choices marked on the ballot are correct from her perspective before inserting the PB in the PBE. This verification guarantees CaI fulfillment, as in the current Swiss RPV procedures. Hence the voter plays a crucial role in the verification process. As the evidence of the cast ballots is publicly accessible, the voter can verify that the published selection shortcodes match the selections marked on her ballot copy. Furthermore, the voter can also verify that the spoilt ballot shortcodes match the spoilt column shortcodes. By performing these two checks [22], the voter ensures that RaC is achieved.

Finally, achieving TaR is more involved: The voter can verify that the final results have been decrypted correctly. However, this is where the automatic verification possibilities reach their limit. If the voter (or any observer) wants to ensure that all votes are represented correctly in the final results, they must inspect the BC themselves. Furthermore, they need to re-run the homomorphic addition of the selected ciphertexts to retrieve the ciphertext representation of the results. These ciphertexts serve as inputs to the decryption proofs, which completes the TaR check and E2E-V. Note that this last step of performing homomorphic addition is deliberately not implemented in another tool accompanying Helverify. The voter would have to trust this implementation the same way she must trust the original implementation for calculating the results. Thus, the only way to verify TaR is to have third parties implement homomorphic addition to verify the decryption proofs. Helverify supports this option by storing the required evidence on the PBB. Finally, EV works the same as in the current Swiss RPV procedures [180] by requiring the voter to sign the VSC, which election officials verify before registering the ballot choices.

Requirements Evaluation

The following section evaluates whether the requirements and constraints defined in Section 6.2.1 are fulfilled by the solution. In terms of Functional Requirements (FR), the main objective was achieving E2E-V, which Helverify fulfills by allowing the voter to use the Voter Frontend to perform the necessary verification steps (FR1). Moreover, verification is optional for voters, allowing them to choose whether they want to verify election procedures (FR2). Furthermore, using Helverify, the VA can manage the life-cycle of an election effortlessly (FR3, FR4). Finally, the voter can cast her ballot by postal mail, as in the current system (FR5).

From a quality standpoint, Helverify also fulfills the specified Non-Functional Requirements (NFR). The system scales for national elections and votes by adapting the principles of the current Swiss RPV procedures, where municipalities and cantons execute the elections and report the results to the federal level [180]. Regarding usability, both the voter and the VA frontend are based on the Material Design principles [119, 213] to ensure an adequate user experience (NFR2). Lastly, by applying HE to ballots, the voter's privacy is protected (NFR3). Furthermore, the association between paper ballots and voters is not stored anywhere, which allows for storing the paper ballots in the VA database without violating BP.

In terms of constraints (C), Helverify is fully compatible with current election procedures in Switzerland (C1), including EV (C2). Furthermore, Helverify augments the current Swiss RPV procedures while leaving the current mechanisms as they used to be (C4). Thus, during system outages, election procedures can continue according to the current procedures (C3). Ultimately, Helverify uses an Ethereum PoA BC and IPFS for hosting the PBB of an election (C5), fulfilling decentralization requirements.

6.2.8 Real-World Scenario

This section explores the hypothetical performance of Helverify by using two real-world voting scenarios' numbers as an empirical foundation. These votes took place in Switzerland in 2021 and are related to the Swiss COVID-19 law [79]. The first vote took place on June 13th, 2021, on the subject of approving a new law regarding COVID-19 measures in Switzerland [38]. The second vote occurred on November 28th, 2021, regarding changes to the same COVID-19 law [35]. For one thing, the reason for choosing these two options lies in the high voter participation [39] due to intense discussions and campaigns at the forefront of both votes [113, 248]. Secondly, both votes were executed on a national level, which allows for using these numbers to project the ballot creation and tallying durations in a realistic, high-volume (in proportion to other Swiss votes) nationwide scenario.

The projection is constructed as follows: The measurements, as shown in Tables 6.3 and 6.4, were used to deduce interpolation functions for ballot creation and tallying. For both real-world votes, the time to create the required number of ballots (*i.e.*, number of eligible voters) and the time to tally the number of cast ballots was calculated for each Swiss canton using the interpolation functions. Thus, the assumption is that each canton would run an instance of Helverify (*i.e.*, the canton is the VA). From a methodological perspective, the interpolation polynomials have been calculated using [324] for ballot creation[1] and [325] tallying[2], yielding the polynomials used for calculating the estimated durations in both scenarios.

The results for the first vote [38] (*cf.*, Table 6.5) indicate that the ballot creation would take between 4 (AI, 12'157 eligible voters) and 480.9 minutes (ZH, 954'327 eligible voters). Furthermore, tallying would take between 30 seconds (AI, 7'919 cast ballots) and 38.9 minutes (ZH, 566'730 cast ballots). Moreover, under the assumption that one federal instance of Helverify was used for all of Switzerland, the roughly estimated durations for creating the ballots (*i.e.*, 5'507'117 ballots) and tallying (*i.e.*, 3'285'326 ballots) amount to 2795.8 minutes and 210.4 minutes, respectively. Note, however, that this total estimate is only a rough indicator for real-world ballot creation and tallying durations, as performance penalties of some degree are likely for large amounts of ballots.

For the second vote [35] (*cf.*, Table 6.6), the results are similar to the first vote: The ballot creation would last between 4 (AI, 12'175 eligible voters) and 480.8 minutes (ZH, 956'792 eligible voters). In this case, the tallying process would take between 30 seconds (AI, 8'615 cast ballots) and 44 minutes (ZH, 643'227 cast ballots). Again, assuming one federal instance for all of Switzerland would yield roughly estimated durations of 2807.2 minutes for creating ballots (*i.e.*, 5'528'244 ballots) and 233.8 minutes for tallying (*i.e.*, 3'633'801 ballots).

Overall, these projected durations indicate that Helverify could handle the scalability and performance requirements regarding the Swiss RPV scenario.

Real-World Suitability While Helverify's performance and scalability are acceptable for municipal and cantonal elections in Switzerland, there is room for improvements. The evaluation revealed the main influence factors and bottlenecks when it comes to the performance and scalability of Helverify. In terms of performance, cryptographic operations greatly influence overall system performance. While at the moment, 2048-bit key lengths are considered secure [15], this might change in the future.

[1] $0.0357181 \quad 0.000321505 \quad x \quad 9.143035291183397 \quad 10^{-10} \quad x^2 \quad 7.5777236271063045 \quad 10^{-16} x^3$

[2] $0.0206561 \quad 0.0000563001 \quad x \quad 4.38172 \quad 10^{-11} \quad x^2 \quad 3.88379 \quad 10^{-17} \quad x^3$

Table 6.5: Projected durations: Vote on Covid 19 law (13.06.2021) [37]

Canton	Eligible Voters	Cast Ballots	Participation	Create Ballots [min]	Tally [min]
AG	435'285	256'371	58.9%	250.6	16.7
AI	12'157	7'919	65.1%	4.0	0.5
AR	38'912	25'665	66.0%	13.8	1.5
BE	743'476	462'982	62.3%	433.0	31.6
BL	189'815	112'652	59.3%	88.8	6.9
BS	114'147	66'301	58.1%	47.4	3.9
FR	209'893	129'670	61.8%	100.7	8.0
GE	271'111	137'841	50.8%	139.2	8.5
GL	26'580	15'774	59.3%	9.1	0.9
GR	140'933	84'068	59.7%	61.3	5.0
JU	54'016	31'610	58.5%	19.9	1.8
LU	281'961	183'604	65.1%	146.3	11.6
NE	113'266	61'157	54.0%	47.0	3.6
NW	31'659	21'605	68.2%	11.0	1.3
OW	27'058	19'328	71.4%	9.3	1.1
SG	327'307	194'989	59.6%	176.6	12.4
SH	52'945	38'565	72.8%	19.4	2.3
SO	181'363	107'135	59.1%	83.8	6.5
SZ	106'765	73'070	68.4%	43.8	4.4
TG	176'200	107'938	61.3%	80.9	6.6
TI	223'768	108'809	48.6%	109.2	6.6
UR	26'880	17'077	63.5%	9.3	1.0
VD	460'172	265'969	57.8%	267.7	17.4
VS	229'071	135'932	59.3%	112.5	8.4
ZG	78'050	52'565	67.3%	30.3	3.1
ZH	954'327	566'730	59.4%	480.9	38.9
Total	5'507'117	3'285'326	59.7%	2795.8	210.4

Helverify allows for using longer keys, but this comes at a performance penalty to the point where cryptography can become a system bottleneck. One option to anticipate this issue would be to refactor the cryptographic primitives to use Elliptic Curve Cryptography (ECC), allowing for shorter keys while ensuring the same security level (*cf.*, security strength comparison of [14]). Another limiting factor is the throughput of the PoA BC: During the evaluation, the block period of 8 seconds proved stable while allowing a decent number of transactions with a block gas limit of 180'000'000. With more fine-tuning, the transactions per second could probably be increased a bit. Finally, the most limiting factor is the number of options/candidates per election, where the duration to create ballots is subject to polynomial growth.

Table 6.6: Projected durations: Vote on changes to Covid 19 law (28.11.2021) [36]

Canton	Eligible Voters	Cast Ballots	Participation	Create Ballots [min]	Tally [min]
AG	437'516	287'482	65.7%	252.2	18.9
AI	12'175	8'615	70.8%	4.0	0.5
AR	39'012	28'425	72.9%	13.9	1.7
BE	744'859	496'811	66.7%	433.6	34.0
BL	190'760	127'723	67.0%	89.3	7.8
BS	113'956	73'682	64.7%	47.4	4.4
FR	211'226	133'306	63.1%	101.5	8.2
GE	273'057	147'460	54.0%	140.5	9.2
GL	26'637	17'616	66.1%	9.2	1.0
GR	141'383	93'112	65.9%	61.6	5.6
JU	54'022	32'780	60.7%	19.9	1.9
LU	282'643	200'131	70.8%	146.8	12.7
NE	113'589	65'787	57.9%	47.2	3.9
NW	31'789	23'889	75.1%	11.1	1.4
OW	27'332	20'600	75.4%	9.4	1.2
SG	328'286	226'148	68.9%	177.2	14.5
SH	53'170	40'372	75.9%	19.5	2.4
SO	182'004	120'868	66.4%	84.2	7.4
SZ	107'265	79'656	74.3%	44.0	4.8
TG	177'080	122'364	69.1%	81.4	7.5
TI	224'027	133'645	59.7%	109.4	8.2
UR	26'864	18'915	70.4%	9.2	1.1
VD	464'365	283'944	61.1%	270.5	18.7
VS	230'387	150'666	65.4%	113.3	9.4
ZG	78'048	56'577	72.5%	30.3	3.3
ZH	956'792	643'227	67.2%	480.8	44.0
Total	5'528'244	3'633'801	65.7%	2807.2	233.8

The conclusion of this evaluation leads to the following recommendation. Helverify should be run on the premises of municipalities or cantons to get the most out of the system. By following this strategy, Helverify achieves nationwide scalability.

E-Voting: A topic that cryptographers like to publish papers about but don't like to see deployed in reality because it's perceived as unacceptably risky.

Jean-Philippe Aumasson, 2021 [11]

7 Decentralized Remote Electronic Voting

Recent years saw an increase in voting systems using public permissionless BCs. Although public and permissionless BCs offer transparency and immutability, permissioned consensus is better suitable for voting systems' requirements because an initial level of trust in authorities is always required. Hence, a permissioned Distributed Ledger (DL) immutably storing the voting system's audit trail satisfies these demands measurably.

As illustrated in Figure 7.1, this thesis proposes three DL-based decentralized REV systems in detail to address the issues outlined in current REV systems. Provotum is a decentralized REV system with three versions developed throughout this PhD thesis. Consequently, the most recent solutions are detailed in the following sections: Section 7.1 describes Provotum 3.0 HE, a HE-based REV system, while Section 7.2 and Section 7.3 describe its extensions. Section 7.4 represents ProvotuMn 3.0, a Mixnet-based alternative. Finally, Section 7.5 Æternum is detailed, a voting system that achieves everlasting privacy and thus does not rely on any computational hardness assumptions, directly addressing the issue of quantum computing looming over contemporary cryptography.

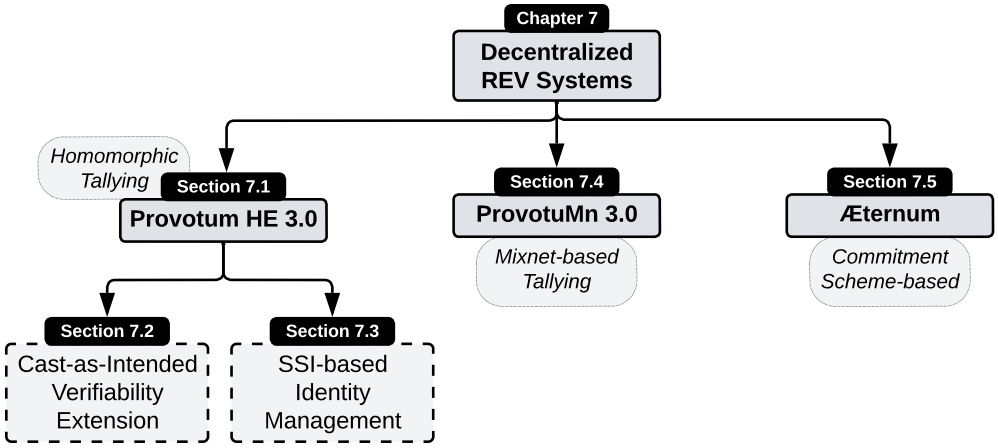


Figure 7.1: Overview of Decentralized REV systems introduced in Chapter 7

7.1 Provotum HE 3.0

As described in Section 4.1.1 The Provotum voting system is a decentralized voting system based on a permissioned DL. The following section describes its high-level design and the details of the voting scheme, followed by the PoC implementation and its evaluation from various angles.

7.1.1 System Design

This section describes the design of the Provotum 3.0 HE system which is depicted in the systems stakeholder Figure 7.2.

The **Identity Provider (IdP)** entity is designed as a TTP responsible for eligibility verification. For authentication, voters present the IdP with their wallet's public (blinded) address. The IdP signs the address and returns the signature to the voter. The voter then submits the signature to the DL, verifying it and storing it. The signature will allow anyone to verify that a ballot, cast from an address, is approved by the IdP. Thanks to the use of Blind Signatures (BS) (*cf.*, Section 2.3.3), the IdP will not be able to link a ballot to the real identity of a voter, hence assuring a secure and privacy-preserving authorization mechanism.

The **Randomizer** entity is introduced to achieve RF in the voting scheme. The Elgamal encryption scheme includes a random nonce chosen at encryption time by the voter's device, it could serve as a receipt, allowing a malicious voter to sell their vote by proving how they voted to a vote buyer. Hence, the Randomizer introduces a blinding factor so the voter cannot reproduce the same ballot published

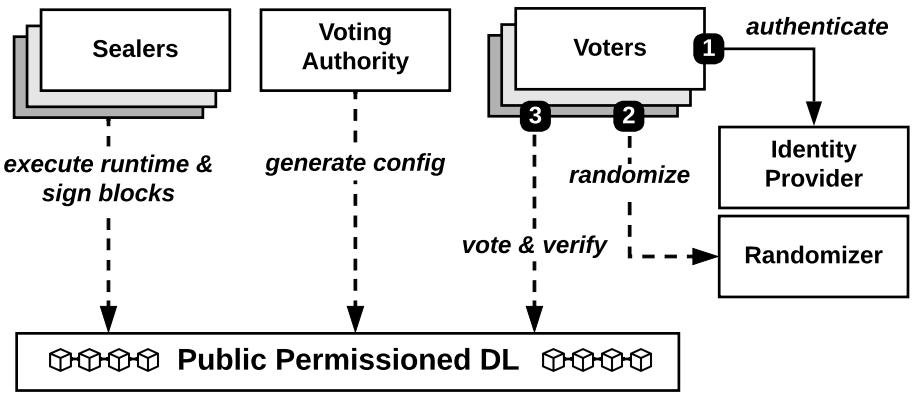


Figure 7.2: Provotum 3.0 HE architecture

on the PBB. The randomizer does not learn anything about the content of the ballots. The randomizer re-encrypts since the blinding (*i.e.*, re-encryption) can be performed on the ciphertext. To enforce that a voter randomizes the ballot, a randomizer's signature will be required to cast a vote. To avoid a SpoF and mitigate censorship by a randomizer, multiple randomizers can be deployed, which could be operated by cantons, organizations, and NGOs. Further, a randomizer can be replaced with a government-issued hardware token. It is important to note, however, that RF would not hold if the randomizer and a vote buyer would collude.

Sealers are entities responsible for executing the DL validator nodes and participating in the DKG phase. Their public key shares is be used to generate the election public key, which will be used by voters to encrypt ballots. When the voting period is over, the sealers collaboratively decrypt the final tally. Due to the distributed nature of votes and elections, each polling place, city, or canton could run a sealer.

The **Voting Authority (VA)** is responsible for coordinating the voting scheme phases: *(i)* The VA coordinates the DL bootstrapping process, coordinating the Sealers, *(ii)* the VA stores the subject of the vote and the election public key originating from the DKG generation. Lastly, *(iii)* the VA closes the vote by signaling to the sealer to start tallying. A municipality, canton, or federal government could operate the VA.

The **Public Permissioned DL** serves as the decentralized PBB. The Provotum voting scheme is directly implemented in the DL (integrated into its runtime), exposing an API with which it can submit transactions and read states tied to the voting process.

Voters are eligible citizens who wish to participate in the democratic process. They communicate directly with the PBB nodes to cast votes, thus removing the need to trust any single entity – *i.e.*, removing any possible SPoFs, and thus introducing censorship resistance.

Lastly, the **general public** is interested in having fair elections and votes, which can verify the data posted on the PBB at any time.

7.1.2 Voting Scheme

The voting scheme or Provotum 3.0 HE is based on CGS97 [65] (*cf.*, Section 4.1.1). A set of N trustees collaboratively generate a public election key, each of the N trustees holding a secret key individually. Each voter encrypts her vote using the combined public key and posts it to the PBB. A vote encodes a *yes* or *no* as either a 1 or a 0, respectively. By homomorphically summing the encrypted votes, we obtain the encrypted amount of *yes* votes. The trustees (*i.e.*, in Provotum, the authorities) jointly decrypt the sum to reveal the plaintext value. By simply subtracting the total amount of *yes* votes from the total amount of votes, the *no* votes are obtained. A special authority, namely the *Randomizer* helps establish RF by adding a source of mandatory randomization, which is not revealed to the voter.

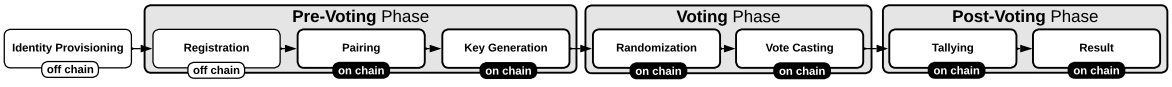


Figure 7.3: Provotum 3.0 scheme's phases

The voting scheme can be divided into three phases (Figure 7.3): *(i)* Pre-Voting/Setup Phase, *(ii)* Voting Phase, and *(iii)* Post-Voting Phase.

A note on the Syntax of Provotum 3.0 HE A ballot consists of a vote v $0\ 1$. $E\ v\ a$ represents the ElGamal encryption of the vote v using a random nonce a. Σ is a DCP proof that the ballot contains either 0 or 1. Choosing a value r uniformly at random from a group of order q is denoted with $r\ _R\ _q$. The homomorphic sum of two ciphertexts is denoted $e_1\ \ e_2$. The multiplication of a scalar c and an encryption $e\ \ e_1\ e_2$ can be computed as $c\ \ e\ \ \ c\ \ e_1\ e_2\ \ \ \ e_1^c\ e_2^c\ \mod p$. H is a cryptographic hash function. In Table 7.1 an overview of the used cryptographic primitives is provided.

Phases	Steps	Primitives
IdM	Voter registration	RSA Blind Signatures
Pre-voting	Registration	ECDSA KeyGen
	Pairing	PoA Consensus Algorithm
	DKG	N N Threshold ElGamal KeyGen, Schnorr
Voting	Ballot preparation	ElGamal, DCP
	Randomization	ElGamal re-encr., DCP, Designated-Verifier Re-Encryption
Post-voting	Tallying	ElGamal, Homomorphic Tallying, Chaum-Pedersen

Table 7.1: Cryptographic primitives applied in Provotum 3.0 HE

Identity Provisioning

The IdP assures eligibility of votes and receives a list of identities from the authorities. In order to register with the system, a voter creates a cryptographic key pair for the DL. The public-key $_v$, is blinded as described in Section 2.3.3. The voter then provides the blinded address $_v$ and credentials to the IdP. The IdP verifies the credentials and signs the address σ `Sign` $_v$. It then returns the signature σ to the voter. The voter now unblinds the payload, revealing the signature for her public-key σ `Unblind` σ When the voter is ready to cast a ballot, she will send the signature σ together with the ballot to the DL. The DL accepts a ballot when the origin $_v$ of the ballot, and the signature σ successfully verify. For the RSA key pair $_{idp}$ $_{idp}$ N d N e and voter's public-key $_v$, we compute the blind signature as follows (all operations mod N):

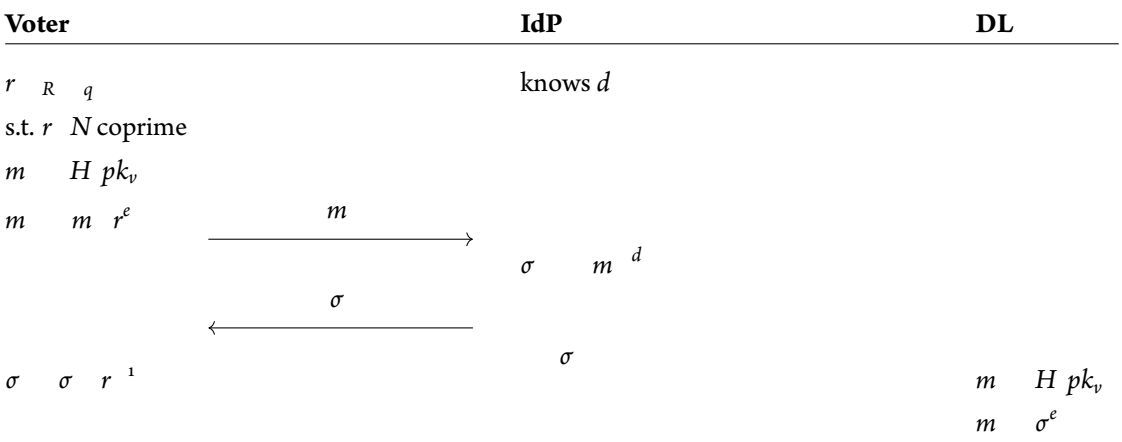


Figure 7.4: Registration of Voters with blind signatures

In theory, the signature σ could be sent at any moment between registration and casting the ballot. For simplicity of the PoC, this is done right after registration.

Pre-Voting Phase

During the Pre-Voting Phase, the Sealers and VA coordinate the creation of the initial DL configuration and the public keys to be used in the vote. In the Registration step, each sealer node generates a DL wallet consisting of a public and private key pair. The address (based on the public key) is sent to the VA, which persists the address. The keys must be transmitted over an authenticated channel. Otherwise, an adversary could impersonate a sealer.

Based on the registered sealers, in the *Pairing* phase, the VA generates a genesis configuration file, which specifies the sealers as the validators of the PoA network, and the VA, which is allowed to create, open, and close votes. The VA publishes the genesis file and starts a DL node to let the other nodes to discover each other. The VA does not act as a validator node, although it could. Each sealer starts its own DL node to create the network and starts sealing blocks.

In contrast with Provotum 2.0, the voting logic is implemented into the DL runtime; thus, no SC deployment is needed. The Substrate chain exposes the necessary API to create a vote, set the public parameters required for encryption, cast ballots, verify proofs, and tally the votes.

The DKG step closes off the pre-voting phase. At the appropriate time, the VA creates a new vote by sending a transaction including the vote's topic and the public parameters in a transaction. Afterward, each sealer fetches the public parameters to create an ElGamal key pair pk_i sk_i and a Schnorr proof. They send the public keys and the proof to the DL. The DL verifies the proof – if it is valid, it stores the public key.

Once all validators have submitted their keys, the VA combines them into a single public-key pk_{voting}, which is stored on the DL and opens the vote.

Voting Phase

The voting phase is divided in two steps. In the first one the voter chooses an option, encrypts it and generates a proof as in Provotum 2.0. In the second phase, the voter interacts with the randomizer to blind the encrypted ballot and to generate a blind DCP proof to provide RF (*cf.*, 7.5) illustrates the second phase.

Ballot Preparation In the first step, the voter constructs a random encryption e E v a , with randomness a, using the vote's public key pk. The voter also generates a proving that v is in

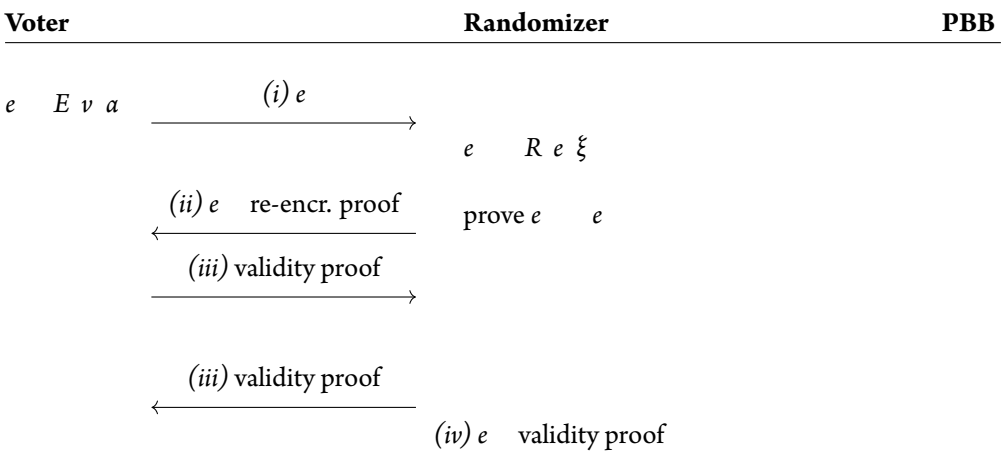


Figure 7.5: *(i)* the voter sends the encryption of her vote v to the randomizer. *(ii)* The randomizer blinds it and proves to the voter that the re-encryption still represents v. *(iii)* Both voter and randomizer interactively generate proof for the re-encrypted vote. *(iv)* The re-encryption and the validity proof are sent to the PBB.

the correct range without leaking v. The second step is an interactive protocol between voters and the randomizer. The voter sends the vote $e \quad E\ v\ a$ and the NIZKP to the randomizer. The randomizer first verifies that the ballot is valid and proceeds with the randomization (*i.e.*, re-encryption) protocol.

Vote Randomization The randomizer can randomize (or blind) the ballot by homomorphically *adding* an encryption of 0, such that the contents of the encrypted ballot do not change [78, 147]. Formally, for a ballot $e \quad E\ v\ a \quad c\ d \quad g^a\ \ h^a g^v$, a randomization e with nonce ξ is defined by the randomization function $R\ e\ \xi$, which can be computed as follows:

$$\begin{aligned} e \quad R\ e\ \xi \quad & E\ 0\ \xi \quad E\ v\ a \\ & g^{\xi}\ \ h^{\xi} \quad g^a\ \ h^a g^v \\ & g^{\xi} g^a\ \ h^{\xi} h^a g^v \\ & g^{\xi\ a}\ \ h^{\xi\ a} g^v \\ & E\ v\ \xi \quad a \end{aligned} \tag{7.1}$$

The randomization performs a re-encryption of v. It is indistinguishable if ξ is chosen randomly under the DDH assumption. Additionally, it is not reproducible if the randomizer behaves honestly and uses a different random parameter ξ for each ballot randomization. To prove to the voter that the returned ballot e does not encrypt a different vote as intended, the randomizer must produce a

ZKP⊠he simplest way to do this is to prove the equivalence of two discrete logarithms as $E(0, a) = e^* \ominus e$. The meaning of $\ominus$ is *Homomorphically add the inverse of the value, i.e.,* $e' \ominus e = e' \oplus e^{-1}$.

Given a voter's ballot $e = E(v, a)$, a randomization $e^* = R(e, \xi) = E(0, \xi) \oplus e$, a public-key pk, and public parameters (p, g, q), we want to prove that $e^- = e^* \ominus e$ equals some encryption of 0, *i.e.,* $e^- = E(0, \xi)$, without leaking ξ [147]. Despite the proof in Figure D.1 in Appendix D, one last step is necessary to ensure RF. If that proof was sent as such to the voter, then the randomization proof, in combination with the original and re-randomized ballots, presents a receipt of the vote. For this reason, Designated-Verifier Proofs with Schnorr's identification scheme are implemented [269].

Given all the parameters from the previous definition, together with the public-key $pk_{voter} = g^{sk_{voter}}$, we wish to prove that $e^- = e^* \ominus e$ is equivalent to some encryption of 0, *i.e.,* $e^- = E(0, \xi)$, without leaking ξ [147]. The algorithm can be turned into a NIZKP via the FS heuristic, with $c = H(e', e^-, t_2)$ (*cf.,* Fig. D.3 in Appendix D). This proof provides the voter with sufficient evidence that her vote has not been altered. However, we still need to prove to the rest of the system that the randomized ballot is correct.

Ballot Proof Randomization

A validity proof for a randomized ballot $e^* = R(E(v, a), \xi)$ is a non-interactive proof, where e^* contains a valid vote $v \in \{0, 1\}$. Neither voter nor randomizer can produce the proof alone, as neither knows all parameters that formed e^*. For this reason, voters and randomizers must collaborate to create a NIZKP which the rest of the system can verify. The algorithm can be divided into two steps: *(i)* voter and randomizer collaborate to create a divertible proof for e [147, 233], then *(ii)* the randomizer diverts this proof into a valid proof for e^* [78, 147]. Given a vote $v \in \{0, 1\}$, a ballot $e = (c, d) = E(v, a)$, a re-encryption $e^* = R(e, \xi)$, a public-key h, and public parameters (p, g, q), we interactively produce a divertible proof of validity as in Figure D.4 in Appendix D.

At this point, the tuple $\Sigma = (A_0, A_1, B_0, B_1, C_0, C_1, S_0, S_1)$ is a valid divertible non-interactive proof for e – *i.e.,* Σ verifies that e is valid, but can also be modified to verify e^* instead. It is also trivial to see that Σ is, in essence, a normal DCP proof created collaboratively by two parties without disclosing their respective secrets.

A diverted proof for the re-encryption e^* can be computed by modifying the response component of Σ. For a given ballot e, a proof $\Sigma = (A_0, A_1, B_0, B_1, C_0, C_1, S_0, S_1)$, a re-encryption nonce ξ, and a re-encryption e^*, the tuple Σ^* defines a valid proof for e^*, with

$$\Sigma^* = (A_0, A_1, B_0, B_1, C_0, C_1, S_0 + C_0\xi, S_1 + C_1\xi) \tag{7.2}$$

The randomizer will thus return Σ instead of Σ to the voter. Proofs of correctness and soundness are given in Hirt [147] and Desmedt and Chaidos [78]. Of special importance for RF is the indistinguishability property. The blinded proof is indistinguishable from a valid proof generated independently. If the parameters Δ_0 Δ_1 k_0 k_1 are chosen uniformly at random, the probability of a dishonest voter linking the diverted to the original proof, is negligible.

It is important to note that only the commitments A_i B_i are hashed in the protocol (*cf.*, Figure D.4 in Appendix D) *i.e.*, the protocol uses a weak Fiat-Shamir transformation. Unfortunately, because the proof is generated for the encryption e, and then diverted to e , the statement (the encryption) cannot be part of the hash. Should it be the case, then after being diverted, the proof would not be correct (as the ciphertext has changed), and the computation of the hash would result in a different challenge being produced. In conclusion, since the randomized ballots are indistinguishable from any ballot the voter could have generated on her own, a voter could choose not to randomize the ballot. To enforce the protocol, the randomizer must provide a digital signature of the ballot e Σ , which the DL verifies before accepting a ballot.

Vote Casting

The voter directly casts to the PBB, avoiding the need to trust any intermediaries. The PBB verifies that the address (from which the ballot was cast) matches the signature of a known voter registered in the voter-registration phase. If it does, and the address was not used to cast any votes yet, it verifies the Σ via the normal DCP proof. If the proof is also valid, the ballot is stored for tallying.

The DL rejects any ballot which is an exact copy of either the ballot or proof, to avoid copying attacks [62]. An adversary could in theory blind the ballot such that it is a copy, but indistinguishable. However, it cannot generate an indistinguishable proof, due to the lack of the nonce used to encrypt the original ballot.

Post-Voting Phase

The VA can close the vote and proceed to the tallying phase. In this phase, each sealer fetches the encrypted ballots from the DL, sums them homomorphically, and produces a decrypted share of the final tally. All N shares are necessary to find the plaintext, so the share does not leak any information. In order to prove to have decrypted correctly, each sealer produces a that the share was decrypted with their respective private-key sk_i belonging to the public-key pk_i, which was submitted to the chain in the DKG phase. Every sealer then submits the decrypted share together with the proof, which the DL can verify and store, if valid.

Once all sealers have submitted their share, the final tally can be revealed. It is important to stress that the threshold encryption scheme requires all N sealers to act honestly. If any sealer acts maliciously, the tally cannot be computed.

The final plaintext value will reveal the amount of *yes* votes, cast for a specific topic. By simple subtraction, the number of *no* votes can be computed:

$$votes_{yes} \qquad votes_{total} \qquad votes_{yes} \tag{7.3}$$

7.1.3 Security Analysis

The following table is a revision of the threats identified in Provotum 2.0 (*cf.*, Tables D.2, D.3, D.4 in Appendix D). The mitigated threats are outlined in the table D.2 in Appendix D.

Ballot Secrecy

Because of the DKG, BS is provided assuming that at least one sealer remains honest since all Sealers are required to decrypt any ballot. The ZKPs of the ballots do not leak any information. Thus, the ballot remains secret.

The secrecy of the ElGamal scheme has already been proven in previous work and will not be further discussed here. However, the randomizer warrants additional elaboration. To achieve BS, it needs to be proven that the randomizer does not learn the contents of any ballots it randomizes.

The randomizer receives the encrypted ballot, two DCP proofs, one wFS, and one sFS. The secrecy of the encrypted ballot relies on the DDH as mentioned [95]. The DCP proofs are NIZKPs and thus leak no information under the random oracle assumption. The randomizer then computes an encryption of 0 and adds it to the original ballot. To do this, no further information is needed on the ballot. Thus, the randomizer learns nothing about the voter's choice.

Correctness

The correctness of the tally is assured with the DCP NIZKP, proving the correct decryption of the values. Since the PBB is publicly readable, anyone can verify the correctness of the proofs. A ballot provably contains only 0 or 1, assuming that the voter and randomizer do not collude. This is because the re-randomized ballot will apply the wFS transformation to yield a diverted proof for the ballot. The wFS heuristic has been shown [27] to be flawed in that a prover can construct a proof that verifies if the value does not lie in the correct range. Assuming either the voter or the Randomizer remains honest, an incorrect ballot will not be recorded on the PBB.

- If a *voter is malicious*, but the *randomizer is honest*: the voter must deliver a ballot proof using the sFS transformation to the randomizer, which it verifies. Thus, the wFS cannot be used in this case.

- If a *randomizer is malicious*, but the *voter is honest*: the Randomizer must produce a proof of correct re-encryption using the sFS. The proof of re-encryption proves to the voter that the ballot still encrypts the original message m, as intended by the voter. So, the randomizer could not construct an invalid ballot and prove to the voter that the re-encryption still represents the intended message m, without the voter noticing it.

As such, the only scenario in which an invalid ballot could be cast is in case of collusion between a randomizer and a voter, which unfortunately raises the trust in the randomizer. Future work could try to derive a divertible proof applying the sFS heuristic instead of wFS.

Receipt-Freeness

To show that the Provotum 3.0 HE system provides RF, the following observations are crucial:

- A voter does not learn the nonce used in the randomization. Due to this, she cannot reconstruct the ballot on the PBB. Proofs that the randomizer produces are all ZKPs and, as such, do not leak the randomization witness: if the voter redid the process, the randomizer would use a new nonce – hence, the randomization of the ballot is not reproducible. Conversely, it is essential to note that if the randomizer and the voter (or the vote buyer) were to collude, RF would not hold anymore. BS, however, remains intact.

- The voter receives a designated-verifier proof of correct re-encryption. This means that the proof is not transferable to a third party for verification: if it were, a voter could send her ballot, the re-encryption proof, and the encrypted ballot on the PBB, and any third party could verify the relation, breaking RF. Using the designated-verifier proof means that the proof makes sense *only* to Alice (the voter), but not to a vote seller.

- The diverted DCP proof of validity is chosen uniformly at random and is thus indistinguishable from the original proof. The initial version of the algorithm used for this thesis did not *fully* render the proof indistinguishable and would thus leak information, essentially breaking RF. This was because the initial draft did not blind the commitments but only the response part of the DCP proof. To achieve RF, the voter must not be able to reproduce any piece of information on the PBB – this includes the proof. Simply blinding the ballot, but being able to

reproduce the proof (which uses the same nonce, among other random parameters employed for the encryption), would indicate that the voter knows the nonce used in the encryption with non-negligible probability. Thus, it would be possible to determine a relation between the proof produced by a voter and her ballot posted on the PBB. While it is not yet entirely proving *how* a voter voted, this might be enough to convince a vote buyer. For this reason, it is essential to blind the proof on the PBB fully.

Coercion-Resistance

The system does not provide CR. Swiss legislation does not permit to cast multiple ballots, which is often mentioned as a way to provide CR. However, since the voter's wallets would be known to coercers, multiple ballots from the same address would not protect a voter from repercussions, which is the main point of CR. Provotum 2.0 [179] also did not provide CR.

Env-to-End Verifiability

For a system to fulfill E2E-V, it has to provide all of its three sub-properties, whereas CaI is not provided in the PoC outlined in this section:

- Cast-as-Intended The system presented in the previous sections does not provide CaI. The only means of auditing the contents of a ballot would be to record the nonce employed in the encryption and use it to reconstruct the pre-randomization ballot. The recreated pre-randomization ballot, together with the proof of randomization, can be used to audit the contents of an encrypted ballot as published on the PBB. Recording the nonce might increase the attack surface for an attacker to learn the contents of the voter's ballot. Many systems advertise possessing CaI under the assumption of an honest client device. However, this seems unjustified in the context of remote voting. It is hard to assume that a voter's personal computer would be properly maintained, patched, and malware-free. Provotum 2.0 [179] was also missing CaI.

- Recorded-as-Cast Once the ballot and proof have been cast; these are stored directly on the BC without going through intermediaries. As such, a voter can easily verify that the ballot was RaC, by checking if the generated ballot and proof match what is stored on the chain.

- Casted-as-Recorded With the information stored on the BC (*i.e.*, ballots and proofs), anyone, including the voters, can verify that the steps were computed correctly. Any attempts at manipulation would be immediately noticeable to any observer.

In conclusion, due to the missing CaI property, the system does not fulfill E2E-V. The extensions outlined in Section 7.2 address the issue of CaI in full, introducing CaI and, subsequently E2E-V to Provotum 3.0 HE.

Eligibility Verifiability

The IdP is responsible for verifying voter eligibility. The BC verifies that each ballot comes from an address, verified through blind signatures provided by an IdP. Thanks to this mechanism, the IdP cannot trace an identity to a wallet address and to a ballot.

The IdP remains, however, a central weakness in the scheme, as a rogue IdP could:

- sign its own wallets to participate in the vote, which could go undetected, as not all registered voters would vote.
- deny signatures to suppress voters, which would be noticeable.

Additionally, a mechanism based on blind signatures provides also concerns on the more practical side. In particular, the following two points are possible examples, both relating to the security of the voter's credentials, but handling two different aspects:

- If a voter's credentials were stolen and used to sign an address and cast a ballot: the voter would notice when requesting a signature, as the IdP would refuse, since it already signed a message for the same voter. The ballot, however, could not be invalidated, since no one, aside from the thief, would know the thief's address.
- Swiss law (VEleS, Art. 4) states: *If the client-sided authentication measure is sent electronically, voters who have not cast their vote electronically must be able to request proof after the electronic voting system is closed and within the statutory appeal deadlines that the system has not registered any vote cast using their client-sided authentication measure.* [81]. The system could provide proof that the credentials were used to register an address, as the IdP could store the blind signature and the identity of a voter. However, definitive proof of whether a vote was effectively cast would not be possible.

Weaknesses of the IdP are addressed in the SSI extensions outlined in Section 7.3, introducing a modular and fully decentralized approach to identity management, directly addressing the weaknesses outlined above.

7.1.4 Performance Analysis

In order to deploy Provotum 3.0 in a real-world scenario, it is necessary to evaluate the system's scalability. First, the theoretical runtime is presented by counting the number of modular exponentiation. Subsequently, a quantitative evaluation of the performance through benchmarks was carried out.

Modular exponentiations (ModExp) are usually computationally the most intense steps of many cryptographic algorithms [129]. Therefore, the performance evaluation of cryptographic schemes is often based on the amount of modExp operations conducted. In particular, the number of operations as a function of votes cast are of interest. Table 7.2, the increased computational cost of Provotum 3.0 compared to Provotum 2.0, for a single election with two choices. Provotum 2.0 had a total of 17 modExp per ballot, while the performance penalty of adding RF accumulates to 26 additional modExp operations. The computation is divided between the randomizer (18 operations) and the voter (8 operations), with no additional overhead to the BC network.

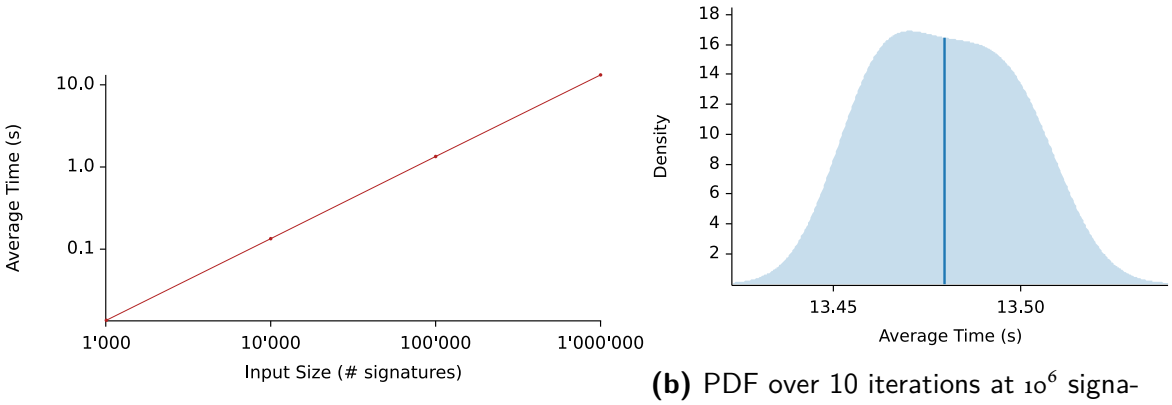
	Provotum 2.0 [179]		Provotum 3.0 HE [148]		
Step	Voter	BC	Voter	BC	Randomizer
Ballot Encryption	3		3		
DCP generation	6		6		
Re-Encr. proof generation					2
Re-Encr. proof verification			2		
DCP randomization			6		16
DCP verification		8		8	
Homomorphic sum					
Total by entity	9	8	17	8	18
Total	17		43		

Table 7.2: Number of modExp operations by entity, in order to cast a single *yes* or *no* vote.

In order to estimate the real-world performance of the algorithms, benchmarks of the voting algorithms are performed. Ideally, the runtime of any protocol should grow linearly with the number of voters and votes cast. To get an idea of the order of magnitude of the tests, we can query past votes in Switzerland to have realistic parameters at hand. The popular vote of the 27th of September 2020, had a voter turnout of around 3'000'000. Thus, the following evaluations take 1'000'000 as a target. As outlined in the security analysis (*cf.*, Section 7.1.3), it would be best to offload calculations off-chain and then publish them on-chain. Thus, the performance of the algorithms is directly evaluated.

The benchmarks were computed by running the given algorithm on an otherwise idle machine. Each algorithm ran at least 10 times and up to 100 times, depending on the runtime of a single iteration. All of the line charts illustrated below use logarithmic scales on both axes, thus, a straight 45-degree slope represents linear growth. The tests ran on a DigitalOcean server on a dedicated 32-core CPU Intel Xeon Gold 6140 CPU, running at 2.30 GHz per core, and 64GB RAM. The total runtime to compute all benchmarks was approximately 315 minutes, while the estimated monthly cost of operation is 640$.

First of all, possible bottlenecks of the Provotum 3.0 HE scheme have to be identified. The Pre-Voting phase, which involves only the set of sealers and the VA, does not need to scale, as this would involve few entities and is not as time-critical. The same holds true for voter registration: The BC is responsible for verifying the signatures of every eligible voter.



(a) Verification of 10^3 to 10^6 RSA signatures

(b) PDF over 10 iterations at 10^6 signatures to verify

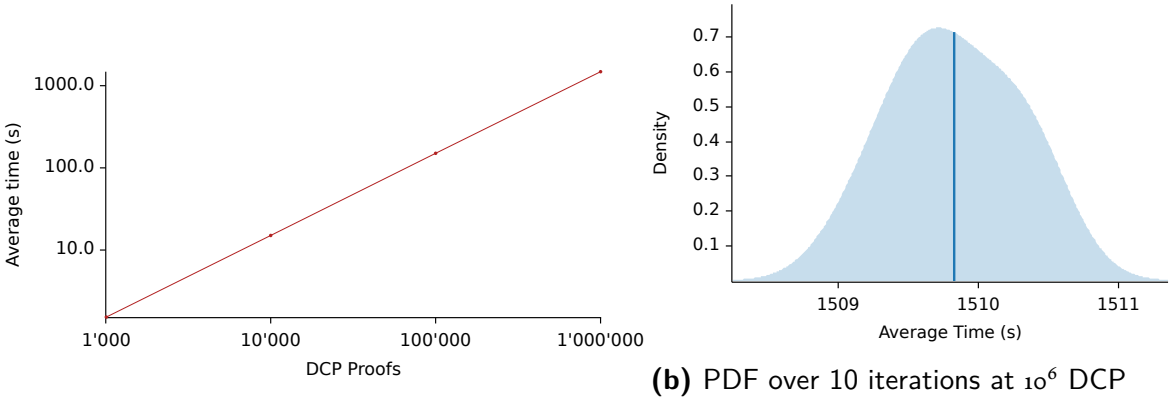
Figure 7.6: Runtime for verifying RSA signatures

The benchmark in 7.6a shows a very clear linear progression and places the runtime of verifying 10^6 signatures at an average of 13 48 seconds, with a throughput of 74 186 elements per second. 7.6b shows the Probability Density Function (PDF) of verifying 10^6 signatures over 10 iterations of the benchmark. The section underneath the curve reflects the probability of any given run achieving a certain runtime.

The Voting phase is composed of *(i)* the ballot generation protocol on the voter's client device, *(ii)* the interactive randomization protocol, and *(iii)* the ballot verification protocol. Of the three, the first only run once on each device. The second protocol is more involved and includes multiple messages sent between the voter and the randomizer. This step needs to scale to allow many voters to participate in the vote. However, this can scale easily by adding multiple load-balanced randomizer

entities or replacing the randomizer server with a hardware token, which would completely eliminate the scalability issue.

The third step is the biggest bottleneck because the PoA BC can be seen as a single unit of computing: due to the Aura consensus algorithm, where a Round Robin approach defines whose turn it is to sign blocks. Thus, the selected authority will need to verify a ballot after receiving it. The BC does not provide any benefit in terms of scalability, as it does not pool its computational resources. For this reason, we further analyze the runtime of the ballot validity proof (DCP) verification.



(a) Verification of 10^3 to 10^6 DCP proofs

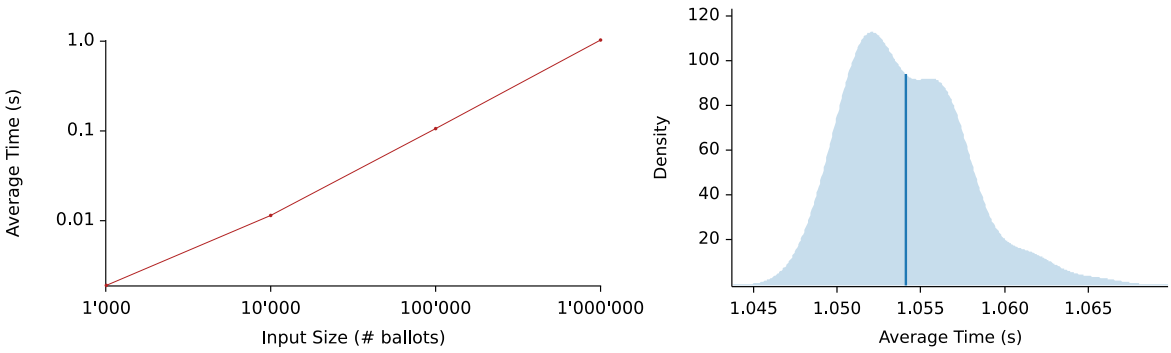
(b) PDF over 10 iterations at 10^6 DCP proofs to verify

Figure 7.7: Runtime for verifying DCP proofs

In Figure 7.7 the runtime for verifying DCP proofs, for an increasing amount of ballots. The benchmark cast 1'000, 10'000, 100'000, and 1'000'000 ballots to verify. The runtime demonstrates a linear growth, with a mean and median runtime of about 1510 seconds and a throughput of 662 elements per second. This is the slowest algorithm to compute. This places it last in terms of speed among all protocols, which is to be expected as the verification of the DCP proofs requires the most modular exponentiations than any other algorithm (*i.e.*, 8 `modExp` per ballot).

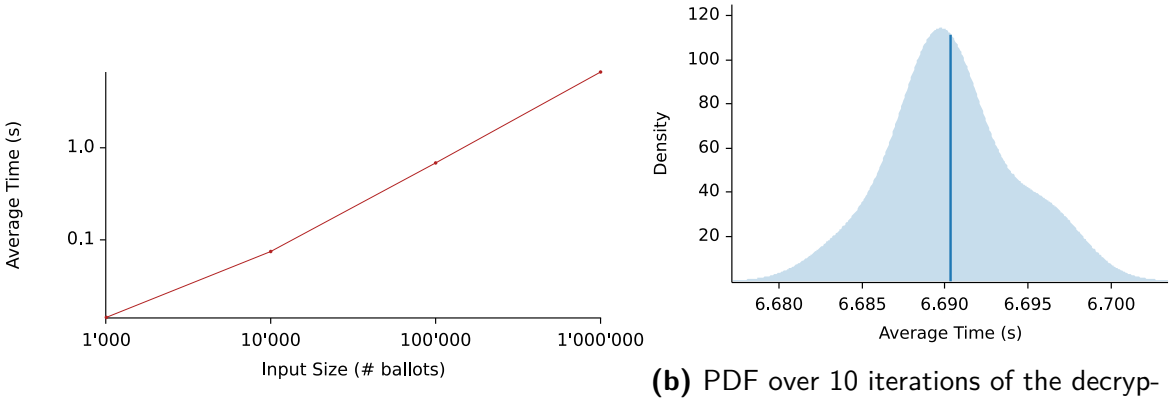
Finally, we have the computation of the final tally in the Post-voting phase. This is composed of the homomorphic sum and decryption of the resulting ciphertext.

7.8 and 7.9 present the runtime of computing the homomorphic sum of up to 10^6 ballots and the final decryption of the tally. Again, they clearly demonstrate a linear growth with the number of ballots to operate on. It took an average of 1.05 seconds to sum up to 10^6 ballots and 6 7 seconds to decrypt the tally, with a throughput of 949 Kelements per second for the sum. As the decryption



(a) Homomorphic sum of 10^3 to 10^6 ballots **(b)** PDF over 100 iterations at 10^6 ballots

Figure 7.8: Runtime for the homomorphic sum of ballots



(a) Decryption of the sum of 10^3 to 10^6 ballots **(b)** PDF over 10 iterations of the decryption of 10^6 ballots

Figure 7.9: Runtime for decrypting the tally of ballots

only operates on a single element (the ciphertext resulting from the sum of ballots), the throughput would not be meaningful in this context. Additionally, the time to decrypt the ballot is directly tied to the number m of *yes* votes, as the algorithm decrypts by brute-force searching for a value i, such that $g^i \quad g^m \mod p$, for $i \quad 0 \; 1 \quad N$, where N is the total amount of ballots cast, $m \quad N$, and g^m is the sum of all ballots. Taking again the results of the past votes as an example, the tally was set at 50%, *i.e.*, half of the ballots were *no* votes, and the other half were *yes* votes. So, the decryption of the sum of 1 000 000 ballots is equivalent to computing $g^i \quad g^m \mod p$ 500 000 times. If all votes were *yes* votes, the decryption would take double the time, while if all were *no*, it would complete immediately on the first attempt.

In conclusion, the benchmarks clearly show that while the system is scalable, the throughput of verifying the validity of the ballots presents a major bottleneck. One possible mitigation would be to separate ballots into three separate storage items: one for incoming, one for verified, and one for spoilt ballots. The system would then process pending ballots and put them in the appropriate storage after verification. The voter would then vote and check at a later time whether the ballot was confirmed as valid. However, this would introduce a time delay and voters would not receive any immediate confirmation of whether the ballot is valid or not.

7.2 Verifiability Extension for Provotum HE 3.0

Resolving the contradiction of privacy and verifiability is the critical element to making REV available to the electorate. A system in which malfunctioning devices are not detected can lead to distorted and fraudulent final results. Optimally, malfunctioning appliances could, besides other things, alter selected votes before encryption, not deliver ballots to the counting entity, or not count well-formed ballots at all.

7.2.1 Verification and Accountability

In REV systems, a voter cannot observe all steps done by an electronic device since the operations on data will not be detected by a voter [185]. A major threat are voter's devices, which can be targets of coercion and malware causing integrity violations. Voters using devices managed by trusted entities of their choosing face fewer problems and threats regarding integrity and malware infections but still face the problem of coercion. To prevent coercion, voters should use devices provided and controlled by the VA. However, if the voting authority is not trustworthy and fair, voters still cannot be ensured that their vote is encrypted and cast as intended [20]. Thus vote verification is an essential allowing voters to verify the electronic voting process and ensure that a ballot is not altered by a malfunctioning voting device [185].

As highlighted in Section 3.1.1, there is an inherent conflict between *verifiability and verification* and *privacy* [2, 158]. On one side, a voter should have as much information as needed to verify that the vote was cast correctly. On the other side, privacy, vote-selling, and coercion become a problem if too much information is provided and a voter can persuade another person about a personal voting selection [2]. Verification and verifiability is in literature also referenced as *voting verifiability*, *election verifiability* or *end-to-end (E2E) verifiability* [2, 175, 185]. Based on [200], E2E-V ensures that the result of a voting round not representing the cast votes is negligible [200] (*cf.*, Section 3.1.1).

On Invidual and Universal Verifiability Furthermore, E2E-V often gets divided into IV and UV [2, 19, 175, 185]. IV ensures that a voter can check that the vote is counted correctly and as intended [19, 110, 185, 283]. Thus, IV is also a combination of CaI and RaC [110, 122, 123, 230]. In [263] IV is defined as the possibility a voter has to check if the ballot reached the voting server and was counted correctly. In some literature (*e.g.*, [198, 263, 283]) IV does not cover the CaI property. The disadvantage of IV is that a voter has to trust other voters that they also verified their vote since a voter cannot check the correct handling of a ballot for others [263]. UV ensures the possibility of publicly verifying that the election outcome coincides with the published ballots [122, 175, 185]. This is the same as CaR [122, 175, 185]. It is important that this can be performed by any voter or an interested third party at a later time [263].

Cortier et al. (2015) prove that IV and UV combined imply end-to-end verifiability under certain assumptions [59]. However, [200] states that combining IV and UV is neither sufficient nor necessary to achieve E2E-V [200]. Since these are opposite statements, the assumptions made by [59] are essential to look at. They assume that ballots cannot be confused, which means that there is a no-clash property [59]. The no-clash property ensures that a voting machine cannot provide the same receipt of a ballot to different voters [199]. The possibility to do this is called a *clash attack* [199]. *Küsters et al. (2017)* do not rely on this assumption [200]. According to them, the combination of IV and UV is insufficient if dishonest voters team up with malfunctioning authorities or devices and cast malformed ballots or if so-called *clash attacks* are conducted [200]. Clash attacks are possible when the *no-clash* property is not assumed, or some entities behave dishonestly and use the applied protocol's weaknesses. The weaknesses are described in the listing below:

- **Devices working together:** If voters do not have to prove that the ballot correctly formulated (*i.e.*, there is a possibility to cast malformed or invalid ballots), voters can fiddle the voting result dramatically [200]. In this case, dishonest voters can cast ballots containing negative or multiple votes of a certain voting option. That will not be detected since only they can individually verify their ballot while it is still possible to tally the ballots stored on the PBB in an universal, verifiable manner. This means that malformed ballots counted since they can only be detected by the voters handing them in [200]. However, the overall verifiability is not valid since the malicious ballots are not detected, and thus, the voting result is not achieved correctly. Overall, individual and UV combined does not correctly work if the voter and the voting device are dishonest and work together [200].

- **Clash attack:** A clash attack is an attack where malfunctioning voting devices or dishonest authorities can cast manipulated ballots since the same receipt is shown to different voters

who selected the same voting options [199, 200]. Thus, malfunctioning devices can deliver the same receipt to similar voters who voted for the same options [199]. An example of a clash attack would be a voting system where the ballots are published on a bulletin board without voter names or pseudonyms attached to it. If in this system malfunctioning devices are present, identical ballots can be created if *n* voters vote for the same options and the same randomness is used to encrypt those. The voting device now only must publish one correct ballot and can manipulate *n - 1* ones at the same time [200]. All voters will verify one single valid ballot (*i.e.* I sincev each voter can validate "his/her" ballot) while not realizing that all of them are verifying the same one. UV is also met because the voting system does not count more ballots than allowed and it is expected that all ballots in the bulletin board are verified by the voters. Thus, the vote of an honest voter was replaced undetectable [200].

In order to achieve CaI, and thus, E2E-V, the following sections outline three different mechanisms for CaI verifiability, followed by a detailed implementation of Challenge-or-Cast verification implemented and integrated in Provotum 3.0 HE.

Challenge-or-Cast Verification Scheme

The major idea of Challenge-or-Cast verification is that voters *(i)* either can cast their ballot or *(ii)* challenge the encryption mechanisms to test if the voting client is honest and trustworthy. It is not possible to cast a ballot and then challenge the encryption of the submitted ballot [19, 122, 141, 195]. This means that all ballots counted in the tally phase were never audited (*i.e., challenged*) [19, 58]. If a casted ballot is also audited, the RF and CR properties will not hold [195]. To ensure RF and CR, a voter needs to restart from the beginning after challenging the ballot's encryption, and the voting device should not recognize the restart after challenging a ballot [58, 122, 195]. There is the possibility of re-encrypting the vote with a newly generated randomness instead of starting from scratch again [19, 122]. Re-encrypting ballots after verification is less preferred since it does not compromise vote secrecy [195]. The process of gaining trust now lies in the number of vote-iterations done by a voter [20, 141, 162, 195].

In this approach, the critical point is that the voting client and the voting system must not know whether a voter is challenging the system or casting the ballot when creating the commitment. A voting device will always generate an immutable *commitment* of the encrypted ballot before knowing whether this ballot will be cast or challenged. This commitment includes the encrypted ballot, which consists of the selected option in an encrypted form [20, 141]. So the commitment contains the selected options but does not reveal their values resulting in no knowledge about the encrypted voting

option [20, 141, 195]. Suppose the voting device does not include the selected voting option in the commitment. The commitment cannot be changed anymore as soon as the voting device knows whether it will be challenged or not. Thus a malicious voting system will be caught if an unselected option is included [20, 141, 195].

If the voting device knows about the voter's challenge-or-cast decision a priori, it can generate a misleading commitment leading to cheating options [20, 20]. In theory, a voter can store the commitment somehow (*e.g.* write it down on paper, copy it to a separate window, scan it with a verifier application) without letting the voting device know where the commitment is stored. Since this commitment is immutable and potentially stored somewhere else, the voting device cannot change it anymore when receiving the voter's intention of challenging or casting the ballot [20, 141, 195]. Thus, if a voter selects *challenge*, a commitment with a not-intended voting option will be challenged, revealing that the voting device cheated and is not trustworthy [20, 141, 195]. This makes it very risky to manipulate a vote if an unbiased voter will challenge the encryption in 50% of the vote-iterations [20, 195].

Challenge-or-cast verification can ensure that a voting client is trustworthy and does not manipulate the selected options. However, above described version does not ensure that the communication between the voting client and the voting system is honest [195].

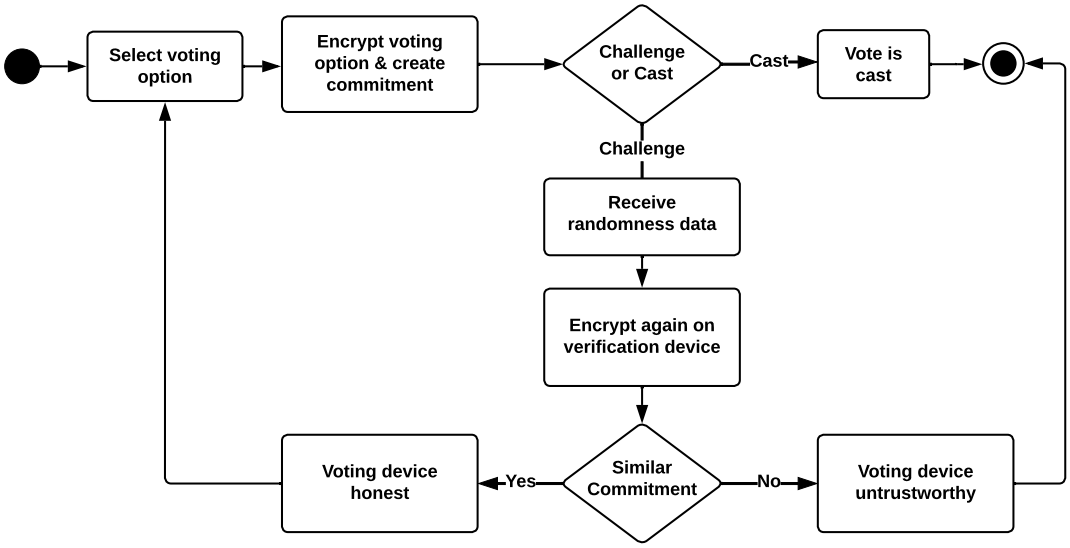


Figure 7.10: The challenge-or-cast process (based on [141])

Figure 7.10 depicts the process of the Challenge-or-Cast approach is illustrated and described hereby (based on [141] and [195]):

1. After a voter successfully logged in, she selects the voting options and then initiates the encryption. Once the encrypted ballot is created, the voting device constructs the commitment. The commitment must include the encrypted ballot, which contains the encrypted voting selections.

2. Once the commitment is created, it will be displayed to the voter. She then should store the commitment somewhere independent and out-of-reach of the voting device. Once this is done, she can either select *Challenge* (to challenge the encryption and thus the trustworthiness of the device) or *Cast* (to cast the ballot and finalize voting).

3. If *Cast* is selected, the encrypted ballot will be sent to the voting system and counted in the tally phase. The voter finished voting.

4. If *Challenge* is selected, the voting device must display the used randomness data in a form a voter can use it for further calculations.

5. Once the voter received all the randomness data used in the encryption, she encrypts the vote on a verification device. This will result in another encrypted ballot, of which again a commitment is created. In theory the two commitments should be exactly the same since the same intended voting options are encrypted, and also the same randomness data is used for the same encryption algorithms.

6. The second commitment can be compared to the first one. If they are similar, the voting device is honest since it encrypted the selected voting options and used them in the commitment. The voter gained trust in the voting device and must restart the process to cast a vote. If the commitment is not similar, the voting device or the verifier device is not trustworthy, and a voter should not vote with this device combination. The commitment can be different if the voting device or the verifier application encrypted not the selected voting options, used wrong randomness data or an error occurred.

Opportunities and Obstacles This verification approach's most significant opportunity is that a voter can challenge the system an unlimited amount of times and until enough confidence in the system is gained [195].

One problem with the challenge-or-cast approach is that the challenge of a ballot requires extra effort from the voter because, after the challenging process, the voting process must start again [271]. This becomes problematic since the possibility to detecting an untrustworthy voting device is only

given when voters challenge the voting device, *i.e.*, willing to make extra effort. If a voter does not challenge the system, she will not detect fraud [195].

Voters may take the extra effort if they do not know a voting system or if they are suspicious about the devices they use. But after multiple successful verifications, the probability that voters rechallenge the same setup may decrease. This potentially results in many people challenging the system in the beginning, but fewer challenges after some votes are already conducted via the system [195]. Since trust is only generated if voters challenge the system this can become problematic [195]. Making the verification mandatory for every new vote would be problematic since the voting device then knows whether a voter has to challenge the ballot or not [195].

Depending on the implementation, two devices are needed for the challenge-or-cast approach [271]. The two devices are preferably in different trust domains, *i.e.*, at least one device is trusted [195]. If the voting device and the verification application / device are not trustworthy, the devices can fake correct encryption and make fraud impossible for a user [195, 271]. Thus, one of the devices must be trustworthy and encrypt the selected option honestly or more than two devices must be used [195, 271].

Lastly, it is counter-intuitive to discard a challenged and potentially successful encrypted vote while casting a ballot not challenged and not verified [195]. Gaining trust via previously challenged ballots can confuse voters, especially if there is not enough context and explanation provided [195].

7.2.2 Comparison of Verification Mechanisms

In Table 7.3, an overview of the *Challenge-or-Cast* (described in Section 7.2.1), *Challenge-and-Cast* (*cf.*, Figure D.6 in Appendix D), *Partial-Audit* (*cf.*, Figure D.5 in Appendix D) and *Code-based* (*cf.*, Figure D.7 in Appendix D) verification mechanisms is given. A key problem is that the voters gain trust in a system as they become more experienced and thus may not take extra effort to verify a ballot despite it belongs to a new poll. This means that voters gained trust in a system over several elections and not in challenging the current voting setup resulting in potentially higher success rates for malicious devices which cast counterfeit ballots [195]. All verification mechanisms which do not implement mandatory verification may suffer from this issue.

7.2.3 Design of Verifier Application

In this section, the design for the verifier application is described based on the process illustrated in Section 7.2.1. Since the challenge-or-cast approach's key point is that the voting device must never know if it will be challenged or not before generating the commitment, an air-gapped solution makes

Table 7.3: Comparison of the Previously Described Vote Verification Mechanisms

	Challenge-or-cast	**Challenge-and-cast**	**Partial-audit**	**Code-based**
Idea	Voter can challenge or cast the ballot with the help of a verifier application	Voter must challenge and cast the ballot and generate simulated proofs to achieve coercion-resistance	Voter verifies certain values (shift or masked vote), but not the vote directly	Voter must challenge the ballots with the help of return codes
Process	After encryption, a voter can either challenge or cast the ballot and thus verify the voting device	After encryption, a voter must insert the trapdoor key such that coercion resistant simulated proofs can be generated	After encryption, a voter verifies the shift or masked vote to reveal the trustworthiness of the voting device	After encryption, a voter must compare and insert certain predefined codes to continue and cast the vote
Needs	Verifier application	Trapdoor key	Basic knowledge in modular arithmetic	Individualized code sheet
Key Point	Voting does not know whether it will be challenged or not	Voting device can create simulated proofs once it knows the trapdoor key	The voter can verify parts of the ballot which do not reveal the selected voting option	Codes must be created and distributed before the voting starts
Advantages	Every voter can create an own level of trust	Challenging and casting of the same ballot	Partial verification of the counted ballot	Verification of connection between voting device and server included
Disadvantages	Verification is optional, counter-intuitive, need for verifier application	Key management for trapdoor key	Only partial verification, selected voting option not verified directly	Need for second channel and management thereof, management of code creation and distribution
Potential problems	Experienced users may not challenge the voting setup every time	Voting devices knowing the trapdoor key before generating the honest proof can work maliciously	Modular arithmetic needs explanation	Second channel for the distribution of the codes may not be existent or appropriate

sense [195]. In this scenario, air-gapped means that the actual voting and the verification are not done on the same device. Furthermore, the voting device should not know which device is used as a verifier so that they cannot communicate before the voting starts. Hence, it would be best if the verifier device can work without an Internet connection such that a voter can decide by himself/herself whether she wants to use the device offline. A smartphone application or a Progressive Web App (PWA) can be used to make this work.

PWAs are enriched web applications that take advantages of new features and technologies into account[270]. They combine multiple advantages of native apps with the ones from websites as they adapt to the given device or browser such that they are almost completely platform-independent

[194, 270]. This can be achieved due to the *service-worker* possibility, which allows native-app-like functionalities and can work as a proxy [270]. Progressive Web Apps have multiple further advantages such as possible installability, offline usage, search engine optimizations, reduced costs, higher speed, smaller size, and no need for a distribution channel such as Apples App Store or Googles Play Store since it can be installed directly through the device's browser [194, 270]. Due to this advantages, in 2017, Gartner listed that in 2020 about half of all native apps were replaced by PWAs [194].

Thus, the decision made was to create a PWA because of the offline usage possibility, platform independence, and easy distribution. Not using an electronic device at all would be very unhandy or even impossible since all the encryption calculations must be done by hand in this case.

Since the verifier should not be connected to the Internet, a way of transferring the data without an Internet connection is needed. Furthermore, the voting device should not know anything about the verifier. Therefore, the following possibilities can be applied:

- **Bluetooth:** Sending data via Bluetooth between devices would have been a pretty obvious choice. However, with his technique, the communication between the voting device and the verifier would not be solely in one direction (voting device verifier) but bidirectional due to the connection establishing process. This harms the feature that the voting device should not know which device is used for verification. A further problem is that voters disabling the Internet connection probably also disable Bluetooth connections simultaneously since they select a smartphone's or tablet's flight mode instead of disabling WLAN and cellular Internet solely. Furthermore, both devices need Bluetooth capability.

- **Near Field Communication:** Basically the same as with Bluetooth, but the unknownness of Near Filed Communication (NFC), as well as the even smaller availability, made an application thereof even less practical. Furthermore, the distance between the voting device and verifier is smaller than with Bluetooth what can become unpractical.

- **Typing by Hand:** This approach would make sense since all communication connections can be disabled on the verifier application. However, typing cryptographic codes and hashes by hand is extraordinarily cumbersome.

- **Camera:** Making use of a camera and some scanning software (*e.g.* to scan Quick-Response (QR) codes or plain text) has the same advantage as typing by hand since it does not need electronic connections at all. Furthermore, if appropriately implemented, only one device needs a camera. The drawback of this approach is that the camera needs a resolution sufficient to scan

displayed data. Scanning QR-Codes is already prevailing in different application areas what simplifies the usage for the voters due to experience.

- **Sound:** Transmitting data by sound waves would need loudspeakers on one device and a microphone on the other. This approach has drawbacks of privacy in regards to other devices listening at the same time. While it is visible which device scans some data with the camera (due to the position in front of the point of interest), it is not possible to easily recognize whether multiple devices listen to sound at the same time or not. Also, there is potentially more noise compared to scanning with the camera. Encrypting the data before sending via sound would bring extra effort and additional complexity.

Since in Switzerland only 2.8% of the population has no smartphone or tablet, the decision was made to use these devices' camera to scan the data in the form of Quick-Response (QR) codes [294]. Furthermore, a tablet could also be used as a voting device especially since the screen of it is large enough to display the QR-code in a sufficient size. Additionally, this technique would allow a complete offline usage with communication only from the voting device to the verifier and not bidirectional.

The QR-codes will contain the data needed for the verification. The content of the QR-codes, based on *Patrick Hayes* implementation in the programming language Rust, is described in the listing below and illustrated in figure 7.11 :

- **Commitment:** The QR-code for the commitment contains an immutable identifier of the encrypted ballot containing the encryption of the selected voting option. This can be achieved with the usage of a secure cryptographic hash of the encrypted ballot. It is essential that the selected option is included since otherwise, the voting device can change it later, and also that the choice is encrypted because otherwise the voter would receive a receipt of the selected voting option. If the selected option is included in the encrypted ballot, creating the hash of a ballot not containing the selected option would end in a different cryptographic hash. Because an honest verifier does not know the randomness data used for encryption at this point, there is no possibility to generate a receipt or try to figure out the input in polynomial time.

- **Challenge:** The challenge, which will only be displayed if the voter wants to challenge the encryption, contains all used data for encryption and the creation of the ballot. This means the selected options are sent to the verifier and the randomness is used for encryption. There is also the possibility of not sending the chosen options to the verifier but letting the user select or insert the voting options again in the verifier. However, especially if polls need text input,

such as elections of people, inserting text on the smartphone may be cumbersome for certain people. Thus, it may be more user-friendly to display the received voting option and let a user confirm those. Furthermore, when verifying the same device multiple times to establish a higher trust level, it is easier to confirm the displayed options than inserting them in every round. This probably raises the likeliness of a verification round due to smaller extra effort. By receiving the commitment, the user gets a fully transparent receipt of the selected voting options what results in the fact that the ballot belonging to the received commitment must never be cast. However, ensuring that there is no possibility to challenge and cast the same ballot must be ensured by the voting device. Also, publicly known data (such as public keys or general data) can be sent with the challenge.

Both QR-codes should contain an ID such that the verifier can quickly determine which QR code is scanned and thus also in which step the voter currently is. This can also help to prevent wrong order scanning since the verifier can give a warning if the challenge is scanned without the commitment scanned before.

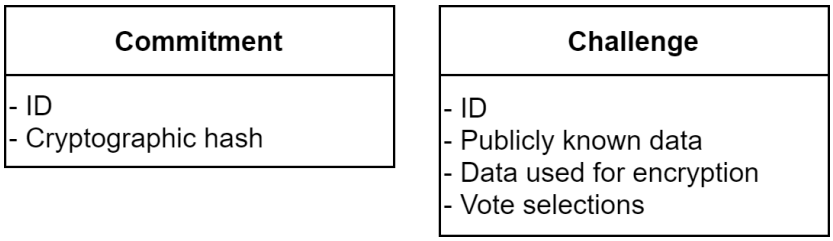


Figure 7.11: The content of the QR-Codes Used for the Commitment and the Challenge (based on [141]).

As soon as the voter scanned the commitment QR code with the verifier application, the verification begins. The scanned commitment must somehow be visible for the voter such that she can compare it to the one shown on the voting device. Comparing a cryptographic hash by eye is not done effortlessly. Thus when displaying the hash to the user, an easily readable format (*e.g.* space after every second character such that tuples emerge) should be chosen. Furthermore, items like cryptographic hash icons can be added to simplify the comparison. However, as soon as some color combinations are used, comparison solely on icons can become impossible for color-blind people. Lastly, the hash (and if used the hash icon) must always be visible during the verification process such that small exchanges of them are easily visible.

When the voter scanned the challenge, she needs to confirm the selected voting options. If a voter does not confirm the selected voting options, the verification process must be stopped immediately.

This because there could be a malfunctioning voting device sending unselected voting options. If the user confirms the shown selections, the verifier encrypts the options with the same encryption algorithms and also the same encryption data. If everything was conducted honestly, this should result in the same encrypted ballot, which can be hashed by the same cryptographic hash function to create the commitment. Once the second commitment is created, the previously received one and the freshly computed one should be displayed to the voter next to each other. Here the same displaying style should be used as before, and maybe even an alert should appear if the two hashes are not similar. However, only displaying that the hashes are similar (or not) without displaying the underlying data is insufficient since then a voter cannot compare the commitments by himself/herself.

Independent of the result of the comparison, the user should be redirected to the same landing page as if starting the verification the first time. This indicates to the user that the verifier application does not know whether the voter verified a vote before or not.

7.2.4 Voter Front End

In this section, the additions needed for the verification steps are provided. The two components (the voter front end, and the verifier application) must not know about each other in terms of which device is used for which step. However, the two entities must be aligned in terms of needed actions, data format, and exchange interface. It is unlikely that a different verifier application not designed for a particular voter front end can be used for verification.

The voter front end must store specific data during its encryption process. Namely, all the information contained in the QR-codes (Figure 7.11) must be recorded during encryption and ballot generation. However, it is important that there is no way an honest voter can reach the data when not selecting *challenge* due to the receipt-freeness property.

When no challenge-or-cast verification mechanism is implemented, the voter could cast a vote directly after selecting the desired properties. However, as the verification gets added, the voter must confirm the selected options such that the voting device knows when to create the commitment. Before this, the encrypted ballot must be created such that the commitment (*i.e.* cryptographic hash of the ballot) can be prepared as well. The selection of an appropriate secure hash function is a trade-off between usability and security. If the hash is too long, it becomes cumbersome to compare them, while it becomes risky if a collision is found too quickly or for too many pre-images the same hash exists. However, the ballot is already encrypted for the hash creating, meaning there is no data leak when using a weaker hash function.

As soon as the hashing is completed, the hash should be presented to the voter. This must be done in the same way as in the verifier such that the comparison effort is minimized. This means if spaces

are inserted after every second character of the displayed hash and a hash icon is used, it must be similarly displayed in the voting system.

While the commitment is shown to the voter, she must have the possibility to either select *challenge* or *cast*. If *cast* is selected, the previously encrypted ballot gets sent to the voting servers and if it passes all tests there, it will be put in the ballot box such that it will be counted in the tallying phase. At the same time, the stored data used during encryption gets discarded unrecoverable. If the user selects *challenge*, the information used for encryption is shown. Here it must be ensured that the ballot cannot be cast as well. There must be no way of doing both, challenge and cast, at the same time or consecutively.

After a successful round of verification, the voter should be returned to the first seen landing page to illustrate that the voter front end does not know the number of verifications done previously. This simply because the voter front end is not allowed to know how many times a voter verified the device since it could make probabilistic assumptions whether it will be challenged again or not.

7.2.5 Trust Boundaries of the Provotum System

A *trust boundary* is an edge where a user's level of trust in an application, a system, or hardware changes. The trust boundaries are mainly existent since specific data is exchanged between different devices, probably not in the voter's custody. Therefore, the level of trust a voter expresses towards an entity differs, resulting in various trust boundaries [153]. In Provotum, four different trust boundaries are present, namely *Distributed Ledger*, *Voting Device*, *Randomizer*, and *Verifier Device*.

The trust boundaries are illustrated in figure 7.12 as light grey boxes with dotted borders and explained in the listing below. The white boxes represent the software part included in the Provotum system, while the arrows amongst them are an abstract representation of data exchanged between the entities. This also indicates the one-way connection and data exchange between the voting front end and the verifier application. Hence the impossibility of a verifier transferring data to the voting software is shown.

- **Distributed Ledger** The Distributed Ledger trust boundary contains the Provotum chain, which is distributed over sealers participating in the current vote. Hence, the trust is not centralized on one entity but distributed over the sealers. This reveals classical blockchain advantages like no need to trust a single entity, immutability, and coercion-resistance.

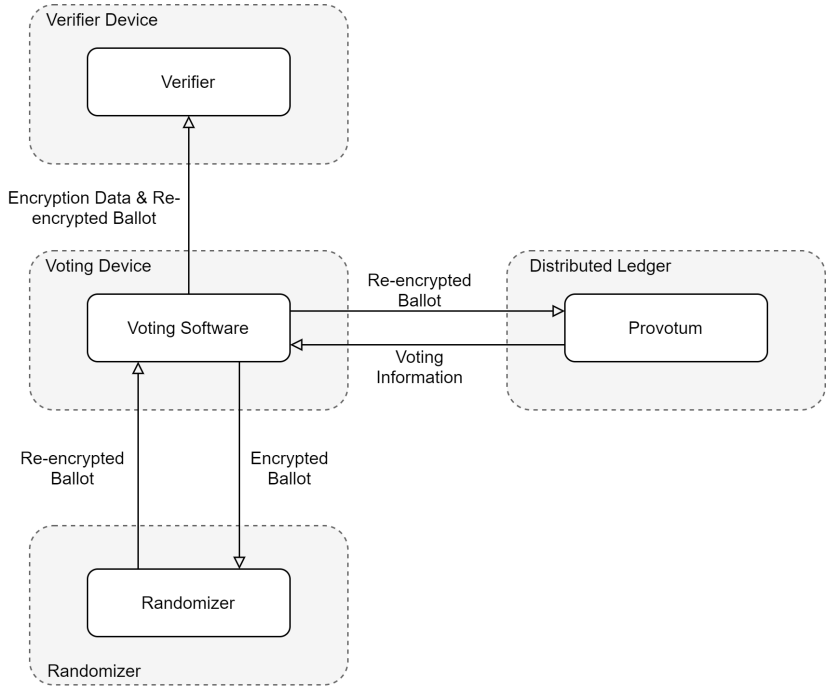


Figure 7.12: Illustration of the Trust Boundaries in Provotum including the Verifier Application.

- **Voting Device** The voting device can be in the voter's trust domain but can also be outside of it. This weakens the restriction of using a voting device that is trusted. If the voting software is not running on a device not trusted by the voter, the trust domain *voting device* is not trustworthy.
- **Randomizer** The Randomizer can be operated by different entities which are not in control of the voter. Hence, the voter may not trust this entity resulting in a different level of trust for this entity compared to others.
- **Verifier Device** The voter should theoretically trust the verifier device. Therefore, it is the only device trusted by the voter. However, it is also possible to use unknown and potentially untrustworthy devices as a verifier.

As visible in figure 7.12, the verifier and the voting software are not in the same trust boundary. The rigid separation of these trust domains allows the usage of devices that the voter may not trust. Since the verifier is used to verify the voting front end's honesty and trustworthiness, especially the latter mentioned can be running in an untrustworthy environment. This allows voters to use devices that are not controlled by a trustworthy organization or not their property. However, it is also possible that the verifier is running in an untrustworthy trust area. In this scenario, multiple devices must be used

as a verifier, and the verification process must be conducted with every verifier for the same input data. A voter can trust the most occurring device assuming the majority is honest and not malfunctioning.

7.2.6 Implementation

Implementing the CaI extensions consists of two major parts: *(i)* the additions to the voter front-end and *(ii)* a new verifier application. The additions to the voter front end are described in Section7.2.6 and contain the creation of QR-codes containing the commitment and the data stored during encryption. Further, the verifier application is explained in Section 7.2.6.

The detailed process is visible in figure 7.13. It is also visible that the verifier application must not be used to hand in a ballot successfully. As Figure 7.12 illustrated, the voting system and the verifier application have different trust domains and therefore a trust boundary exists between them. In Figure 7.13, it is visible that encryption and commitment creation are performed on different devices and thus, also in different trust domains.

Addition to the Voter Front End

The extensions of the React front-end are based on [148]. As described in Section 7.2.6, voters must confirm their selection to initiate the encryption process. To achieve this, an additional step must be implemented such that the ballot creation is not bound to the casting mechanism. Therefore, the initial encryption process is within its function (*i.e.* `export const createEncryptedBallot = (vote, keyring) => async (dispatch) => {...}` in *voter/src/redux/action.js*).

Consequently, two additional functions are called *castBallot* and *challengeBallot* to hand in a ballot and challenge encryption. This minor change allowed the division between the encryption of a ballot and the casting or challenging thereof. Furthermore, it brings the possibility of creating the commitment hash of the encrypted ballot. In Provotum, the encryption of a ballot is based on the encryption of the given answer to a particular voting subject (*i.e.* the answer to a specific voting question). Hence the handed-in ballot is a combination of the encrypted ones for every vote subject.

The voter front-end must store specific data during encryption so that this can be transferred to the verifier with the QR-code help. The data used to verify the encryption in Provotum is listed below:

- `publicKey` contains the `publicKeyH` as well as the parameters `g` (generator), `p` (prime number) and `q` (prime number).
- `voterPublicKey.h` is the voter's specific public key.

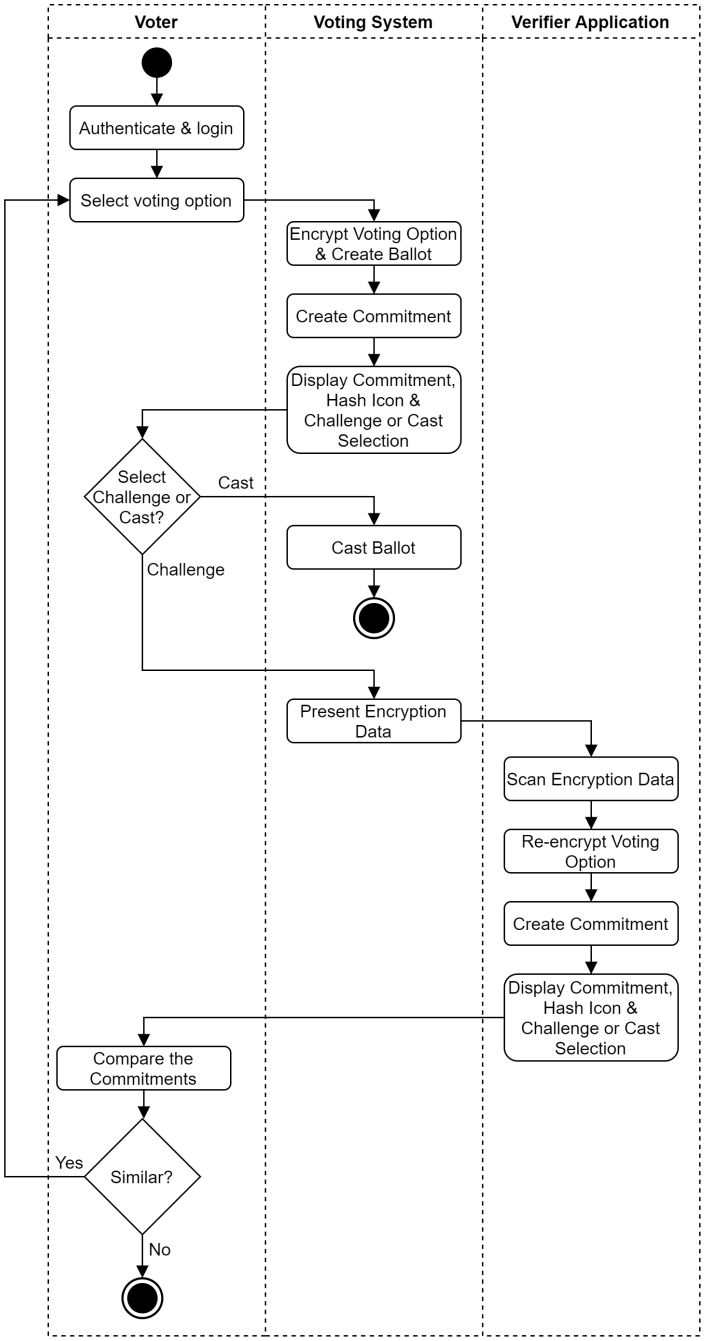


Figure 7.13: The Challenge-Or-Cast Process in Detail

- `answerBin` representing the selected answer in binary form (*i.e 1* for *YES* and *0* if the user selected *NO*).
- `Nonce` is a random integer in the range of the parameter `q` defined in the publicKey's params.
- `reEncryptedBallot` represents the ballot after it is randomized with the randomizer.

- `reEncryptionProof` is the proof used to witness that the re-encryption by the randomizer is done correctly.

Figure 7.14 illustrates Provotum-specific adaptions to the content the QR-code must contain according to Figure 7.11. The commitment includes the voting questions in plain text since they must also be transferred to the verifier device. The `publicKey` and the `voterPublicKey.h`, summarized as *General Data* in figure 7.14, are similar for all voting subjects. Hence, they will not alter during the encryption of different subjects. On the other side, the remaining variables are different for every subject, meaning if multiple voting subjects are included, multiple (*e.g*) *answerBin* are included in the QR-codes. This means, the points *Vote Selection* and *Data used for encryption* in figure 7.14 are included multiple times when multiple voting questions are present.

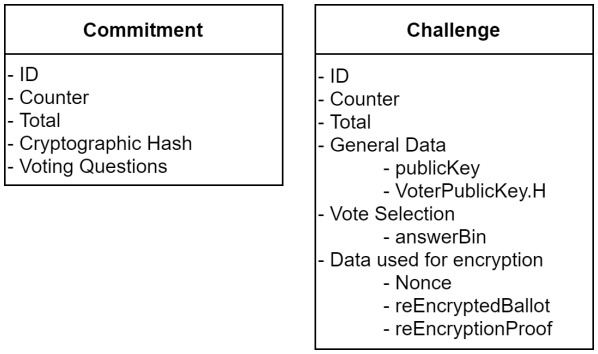


Figure 7.14: The Content of the QR-codes Used for the Commitment and the Challenge in Provotum.

The data listed above is stored in JavaScript objects during the encryption process because this allows easy generating of structures and dependencies. Furthermore, it will enable straightforward parsing to the *JSON* format to generate the commitment and the QR codes. The data is stored in a *Redux* store to simplify the exchange between different React views.

The hash of the ballot (*i.e.* the commitment) is created with an SHA-256 hash of the JSON-element *preparedBallotsForHash* as visible in listing 1. Creating the hash in an isolated step allows simple security enhancements since exchanging the SHA-256 algorithm with a more secure one does not influence other parts of the code. The JSON element used contains the *subjectID* (*i.e.* the ID of the vote question, represented as *sID*), the *encrypted ballot* (*i.e.* the encrypted ballot, which can be accessed through the *encryptedBallotArchive[index]*) and the *cipherToSubstrate* (*i.e.* the data sent to Substrate and thus revealed on the PBB, represented as *cTS*)). An explanation of why this is sufficient is given in Sec 7.2.6.

Once the cryptographic hash is created (*i.e.* after the voter selected *Encrypt Ballot* on the voting device and thus confirmed the selected voting options), the commitment must be shown to the voter immediately. This is done in *voter/src/router/ScreenBallot.jsx*, where the cryptographic hash is fetched from the Redux store.

```
var preparedBallotsForHash = encryptedBallots.map(([sID, cTS], index) => {
    return [sID, cTS, encryptedBallotArchive[index]]
});

const ballotHash = sha256(JSON.stringify(preparedBallotsForHash)).toString();
```

Listing 1: Creation of the commitment

The screen presented to the voter after the ballot creation phase is shown in figure 7.15. As it is visible, at this point, the user can either select *cast* or *challenge* but not both. Furthermore, a voter should scan the QR-code independent of the next step. A pop-up message to summon the user to scan the commitment is excluded on purpose since it is an extra effort for an experienced user to close this message. Allowing the selection between *cast* or *challenge* only after the user scanned the commitment is technically impossible since the voting device does not know of the scanning progress. If the voter selects *cast*, the ballot gets cast to the bulletin board and, if all proofs are correct, gets counted in the tally. If the voter selects *challenge*, the previously stored encryption data must be presented to the voter such that it is possible to scan it with the verifier.

In theory, the challenge's content is similar to the commitment shown in listing 8. The *id* is exchanged with *Challenge* and there is no ballot hash included but a *Key* property. This key contains either the value *GeneralData*, as visible in listing 9, or the *subjectID*, as visible in the code snippets 10 and 11. The verifier assorts the received data according to this key. The *Counter* is needed for the challenge since the data needed for verification exceeds the maximum size of a QR-code.

A QR-Code with an error correction level low and 177 177 modules can maximally contain 4296 alphanumeric characters [168]. Additionally, huge QR-codes are less handy to scan since it needs a better camera resolution as well as a display with higher resolution to display them. Thus, the challenge data is divided into several QR-codes, where the total amount of QR-codes is denoted with the *Total* property and the current QR-code is labeled with the *Counter*. In the current implementation, the first QR-code (*i.e.* the QR-code with *Counter 0* - listing 9) is always used for the general data. Each subject-specific question needs two additional QR-codes (*i.e.* the QR-codes with *Counter 1* & *2* - listing 10 & 11) containing all the voting question specific information. As listed above, this is the

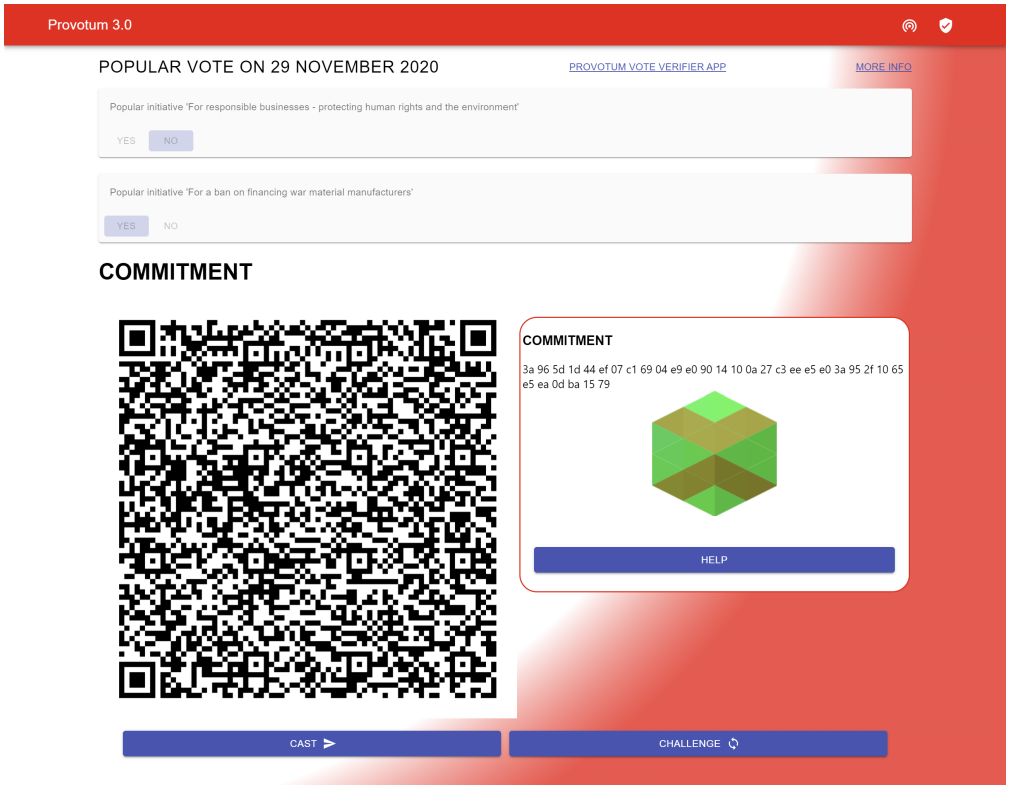


Figure 7.15: The Commitment is Shown to the Voter in the Voting Front-End

answerBin, *Nonce*, *reEncryptedBallot* and the *reEncryptionProof*. In the listings 9 to 11, long cryptographic hashes are shortened, which is indicated by ellipses, due to readability reasons.

The QR-codes for the challenge are displayed in the same manner as the QR-code for the commitment, as visible in figure D.8. The difference is that every second one of the challenge QR-codes is shown in a round-robin fashion. This rotation will be stopped once the voter selected *Back to Start* since the voting device has again no knowledge whether the voter scanned all QR-codes or not. This means that for a total of 5 QR-codes, the voter must, with an optimal scanning success rate, at least scan QR codes for 5 seconds. Besides that, the commitment and the cryptographic hash icon must be displayed all the time such that a voter can recognize potential changes immediately.

Vote Verifier App

As described in the *Design* chapter 7.4.2, the challenge-or-cast approach with the help of QR codes for the data exchange was selected. Furthermore, the verifier application must be able to run offline, which can be achieved by a Progressive Web App. There should be no knowledge of what device is challenging which voting device.

Similar to the voter front end, the user must either select *cast* or *challenge*. If the user selects *cast*, information is shown that she must also select *cast* on the voting device since, otherwise, the ballot will not be cast. Selecting *challenge* will start the verification process. Thus, the scanner will open again, and once the user selected *challenge* on the voting device, she must scan all QR codes displayed by the front end. The scanning progress is indicated next to the scanner, and once all QR codes are scanned successfully, the application will continue automatically. For simplicity, a user cannot select which QR code to scan but has to scan them in the given order. If one QR code cannot be scanned initially, a user must wait until the missed QR code is displayed again. This occurrence is visible in figure 7.20 where the QR-codes with *Counter 0*, *1* and *3* are already scanned while the ones with *Counter 2* and *4* are not done yet.

Scanning a malformed QR-code, scanning the commitment *after* the challenge, or scanning the challenge *before* the commitment, will all result in error messages as visible in Figure 7.21. The error messages are individual to match the fault as close as possible.

Once the challenge is scanned successfully, the application will automatically proceed to the *confirmation* step. In this step, the user must confirm if the displayed selections represent the ones she chose during voting on the voting device. If confirmed, the vote gets encrypted on the verifier device, and the cryptographic hash gets created such that it can be compared against the one shown on the voting device. The voter can compare the received commitment with the one calculated on the verifier device, as they are displayed next to each other. For simplicity, the verifier application shows the success depending on the result (visible in figures 7.23 and 7.24). Furthermore, a voter should compare the commitment displayed on the voting device with the calculated hash. However, when this is already done in the step illustrated in figure 7.19 and no change in either the cryptographic hash or the hash icon is detected during the process, comparing to the voting device is already done.

The encryption and verification on the verifier device are done with the same cryptographic library used in the voter front end since the verification needs the same parameters. (The node module used can be found here `https://www.npmjs.com/package/@hoal/evote-crypto-ts`). The complete encryption and verification process is shown in listing 12. This function takes the array `votingQuestions`, the `publicKey` and the `voterPublicKeyH` as input and returns the encrypted ballots, and a boolean `verifies`, which indicate if all votes are verified or not. The `votingQuestions` array contains all the data received via QR-code.

The verification is also done on a voting-question level. Hence an encrypted ballot is created based on the `encrypt(...)` function implemented in the library *@hoal/evote-crypto-ts*. This encrypted

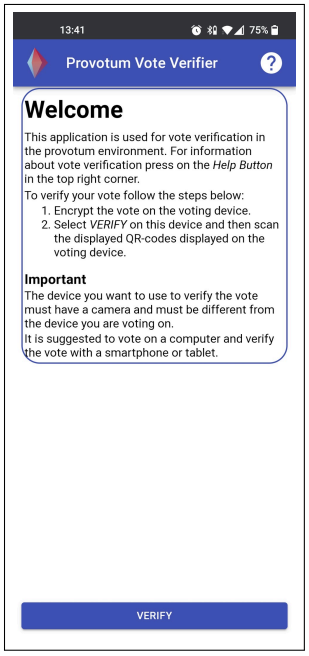


Figure 7.16: Verifier Intro Page



Figure 7.17: Verifier Intro Page with Help

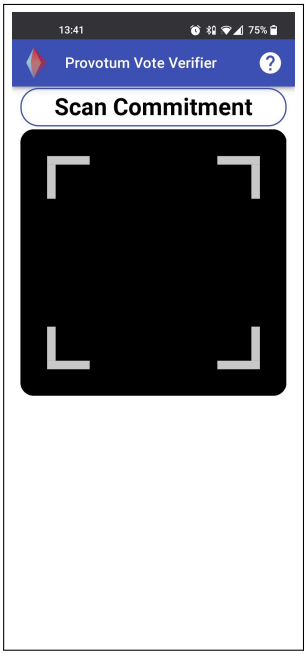


Figure 7.18: Commitment Scanner

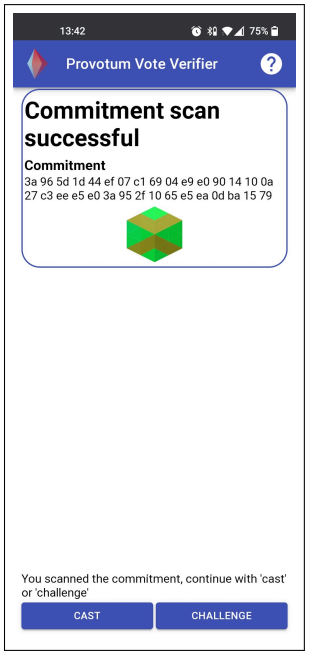


Figure 7.19: Successful Commitment Scan

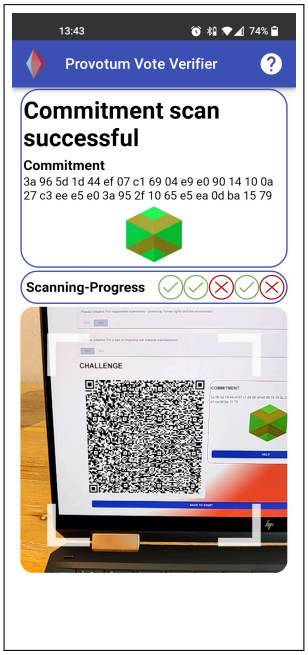


Figure 7.20: Scanning of the Challenge



Figure 7.21: Scanning Error

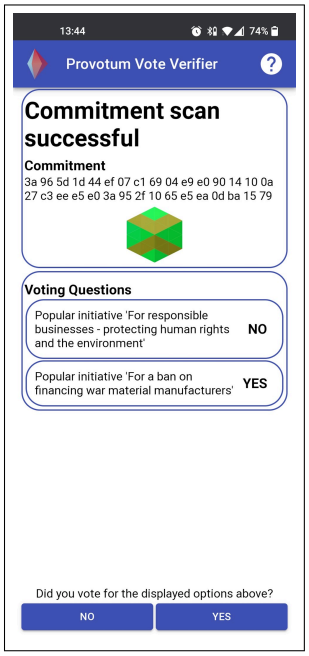


Figure 7.22: Confirmation of Selection

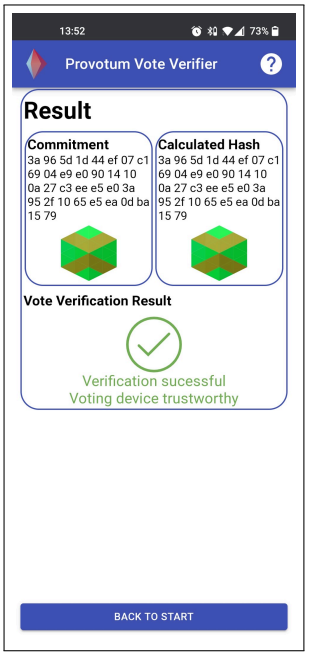


Figure 7.23: Successful Verification

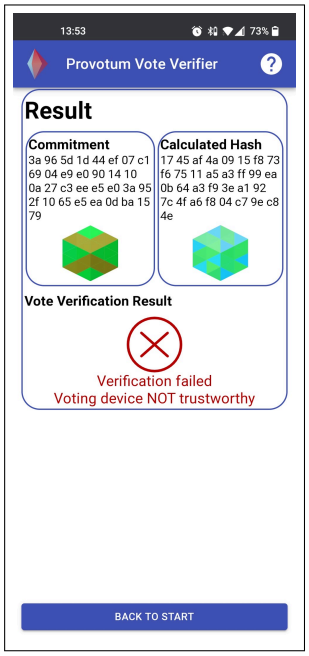


Figure 7.24: Unsuccessful Verification

value is then compared against the value of the re-encrypted ballot, meaning the randomized ballot. The verification is only successful if the values of `encryptedBallot` and `reEncryptedBallot` match each other. While the voting device encrypts the vote, then randomizes and re-encrypts it with the help of the randomizer and then compares the re-encryption with the initially encrypted ballot, the verifier compares the received re-encrypted ballot with the self-encrypted ballot. Since this verification will only succeed if the encrypted Ballot created on the verifier and the corresponding one created on the voting device are similar, it can be verified that the encryption was honest and also that the re-encryption matches the encrypted Ballot. Suppose the verification is successful `verifies` will be *true*.

The value of `allVerified` (listing 12) will only become *true* if, in all verification rounds, the value of `verifies` is *true*. Once one value is *false*, it will be *false* for all remaining rounds of encryption.

With this approach, the randomizer gets verified indirectly since if the randomizer returned wrong values, they will be detected at this point. This allows the verification of the randomizer without receiving random values used for the re-encryption or communicating to the randomizer during the verification process, what allows the verifier to stay completely offline while verifying not only the voting device but also the randomizer.

The creation of the calculated hash, which must be similar to the received commitment to verify the voting setup's trustworthiness, must be calculated with the same cryptographic hash function. Since a SHA-256 hash was used on the voter front end, this must also be used in the verifier. Furthermore, the same input in the same format must be hashed but with the difference that the ballot encrypted on the verifier must be used. If this is different, the verification of the re-encrypted ballot fails and it will result in a different cryptographic hash.

7.2.7 Analysis and Evaluation

In this section, the CaI extension's various dimensions are analyzed (Security, Technical, Process, and Environment). Then the extension is evaluated in comparison to the prior Provotum 3.0 HE and other voting systems.

Security Analysis

First, in Table 7.4, an overview of various threats, including a STRIDE classification, is provided. The mitigation strategies against those threats are summarized in Table 7.5. The ID refers to the inherent threat, meaning threat *T1* can be mitigated with mitigation strategy *M1*.

The STRIDE methodology was brought up by Loren Kohnfelder and Praerit Garg in 1999 [183]. The name STRIDE stands for and is an abbreviation of Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, and Elevation of Privilege [275].

What generally holds true is that the introduction of the second device, the verifier, allows for challenging the trustworthiness of the encryption of the ballot. Therefore, it is not mandatory to trust the voting device blindly but challenge it to establish trust on experience (Threat *T1* in Table 7.4). This introduction mitigates the threat of voting with a malfunctioning or dishonest voting device (Mitigation strategy *M1* in Table 7.5). The threat is classified as *Spoofing* and *Tampering* since specific devices or parts of the system can be manipulated, malware could be inserted or devices could fake their identity.

If the user uses a dishonest verifier, she will likely never cast a ballot, although the voting device is encrypting the ballot correctly. Having an untrustworthy voting device can be detected when exchanging the verifier application. If the verification then fails, the voting device is dishonest with a high probability. It is essential to state that this technique does not ensure that there is never an unselected voting option reaching the tallying phase since it depends on the user's decision whether to challenge the device or not. Suppose the device, respectively, the ballot it creates, is not challenged.

Table 7.4: Summary of the Security Threats including STRIDE Categorization

ID	Title	Threat	STRIDE
T1	Untrustworthy device	The voting device is not trusted. Therefore, voting with it is risky.	Spoofing, Tampering
T2	Insecure connection	Insecure connections such as *HTTP* allow man-in-the-middle attacks.	Spoofing, Tampering
T3	Cross-Site-Scripting	Classic cross-site-scripting attacks to get sensible data.	Tampering
T4	Malicious service workers	The injection of malicious service workers can be achieved on the *localhost* development environment and with the help of social engineering.	Tampering
T5	Entry point to local data	The PWA, especially when installed, can be used by attackers to reach data stored on the verifier device.	Tampering
T6	Denial-of-Service attacks	DoS attacks can target entities providing the verification application such that voters cannot verify their ballots. In combination with malfunctioning voting devices this can facilitate counterfeit ballots.	Denial-of-Service
T7	Coercion-resistance	Voters can become victims of coercion.	Information Disclosure
T8	Deployment of the verifier	The deployment of the verifier can be problematic, if it is only done by a malicious entity.	Tampering, Elevation of Privilege, Repudiation
T9	Leak sensitive data	Attackers try to receive sensitive data.	Tampering, Information Disclosure
T10	Collusion of devices	Voting device and verifier arrange the creation of counterfeit ballots.	Tampering, Information Disclosure
T11	Social engineering attacks	Attackers influence voters not to use the verifier and at the same time provide malicious voting devices.	Tampering

In that case, there is no mitigation against casting a counterfeit ballot since it is technically impossible to detect. Therefore, the voter must challenge and verify their voting setup before casting a valid ballot. It is crucial to motivate users to verify their voting setup every time they use it.

Technical Analysis

From a technical perspective, a PWA is basically an enriched web application. Therefore, attack scenarios and vectors known for web applications are also transferrable to PWAs [270]. However, since PWAs use new technologies and features, some attack vectors can be mitigated before becoming a threat. The so-called *service-workers* are a crucial point of a PWA since they allow app-like functionalities. The *service-workers* only work via *HTTPS* connection, which is crucial to mitigate MiTM attacks [270]. With a *HTTPS* connection it can be ensured, that the *service-workers* are not tampered with on the way from the hosting server to the verifier device [270]. Hence the STRIDE catego-

Table 7.5: Summary of the Mitigation Strategies according to the Threats Defined in Table 7.4

ID	Mitigation
M1	Due to the challenge-and-cast setup, two or more devices can be used. This allows to challenge the voting device which can be caught cheating this way.
M2	PWAs enforce the use of *HTTPS*. Furthermore, there is no communication from the verifier application with a external server containing sensitive data.
M3	Since the data received by the verifier is not sensitive, the intention of an attack is not truly given.
M4	This occurs on *localhost* only. Hence, not using *localhost* for verification and an official version of the verifier instead mitigates this threat.
M5	PWAs run in a browser sandbox even when they are installed on the device. Furthermore, service workers only have access to the cache storage and not to the local or session storage.
M6	There are three main methods against this threat which are preferably combined: *(i)* DoS mitigation tools should be applied at the entity providing the verifier application, *(ii)* the users should install the PWA on their device such that offline usage is possible, and *(iii)* the verifier application should be hosted by multiple entities.
M7	This is only partly mitigated. An honest voter will not receive a receipt, but a dishonest voter can generate a receipt by reading out data at runtime. Furthermore, a coercer observing a voter will see the same information as a voter. This could be fought by allowing to cast multiple ballots and only count the last one.
M8	Multiple deployments or deployment by a Trusted Third Party can mitigate this.
M9	This is mitigated since the verifier does not handle sensitive data since the received data is never cast. Furthermore, offline usage can help against this threat.
M10	This theoretically is possible, but since the voting device and the verifier must communicate, it becomes problematic when using the verifier offline. The unawareness about which device is used as a verifier complicates communication even more.
M11	This can be mitigated technically and processual when making verification mandatory. With challenge-or-cast, this is not reachable. Therefore, active campaigns motivating voters to use the verifier should be applied.

rization is *Spoofing* for man-in-the-middle attacks and *Tampering* due to the possibility of inserting malicious service workers. This threat is summarized as *T2* in table 7.4 and can be mitigated with the strategy *M2*.

However, even with *HTTPS* enabled, there are attack vectors how attackers can install malicious *service-workers* [270]. In the first one, cross-site-scripting and the upload of a use case-specific file are needed, and in the second one, a victim gets forced to install malicious *service-workers* on *localhost* with the help of a social engineering attack [270]. To prevent damage, service workers only have access to the cache storage but not to the local or session storage [270]. Since the user cannot upload a file within the PWA, the first attack vector is not possible. The second one does affect only the *localhost* since *localhost* is excluded from the *HTTPS* service-worker installation policy due to the simpler development of PWAs. Consequently, not running the verifier in production over *localhost* will resolve this threat. Furthermore, PWAs always run in the browser sandbox even if installed on a device [270]. A native app would be verified by the respective App Store reducing the possibility of a malware-infected application. Since PWAs are hosted like normal web pages, that's not the case.

However, PWAs have the advantage that the content update is more straightforward, since the only thing needed is to host a newer version of the application [270].

Denial-of-Service (DoS) attacks against the entity providing the verification application such that the voters cannot access the verifier and thus cannot verify their votes can occur (threat *T6*). However, if the voting device is trustworthy and does not malfunction, the result will not be influenced when a voter cannot verify the ballot. Here, common DoS mitigation techniques need to be applied by the provider, or the application must be provided by various entities. Distributing the provisioning of the verifier would make it more difficult for the verifier and the voting device to work together, since there are more possibilities of combinations (*cf.*, Section 7.2.7).

CR and RF are only valid when an honest voter is assumed (threat *T7*). Attacks on CR and RF are is classified as *Information Disclosure* because these threats would harm the vote secrecy property. Due to the nature of Provotum, a technically adept dishonest user can read out the encryption values at runtime without much effort. This would allow generating a receipt by re-encrypting all possible voting options with the same randomness and then comparing it against the re-encrypted ballot received from the randomizer. If this comparison reveals that the re-encrypted ballot and the newly created one belong together, a dishonest voter has created a receipt. This cannot be mitigated with the current setup of Provotum.

For the implementation, open-source libraries used for the generation of the QR codes and the scanning of those are applied. Furthermore, a library containing all the cryptographic functions tailored for Provotum, the popular Redux library, and public frameworks and templates are used. Since about 7 out of 10 applications face security risks emerging from open source libraries, the selection of these libraries must be conducted carefully [272]. The most crucial point here is that the cryptographic library gets adapted in the voting front end and in the verifier once it gets updated. Otherwise, encryption may result in different results. Furthermore, the hashing with SHA-256 to create the commitment is currently considered secure enough, but it may must be replaced in the future. However, it should be kept in mind that the hash must be easily comparable by eye, which is a limiting factor in regards to its length. Also, an already encrypted ballot is hashed. Regarding the hash icons, it could become problematic if the hash icons representing the commitment are not different enough. But as they are only additional to the alphanumeric hash values presented, comparing the character strings instead of solely comparing the hash icons is the user's responsibility as soon as the comparison is intricate.

Overall the PWA is technically implemented in a way that it does not use any sensitive data for verification (Threat *T9* in Table 7.4). The threat of revealing sensitive data would be classified as *Information Disclosure* since it reveals data to somebody not allowed to see it and *Tampering* since the

source code or data storage needs to be manipulated beforehand. Technically, malfunctioning PWAs need an untrustworthy companion voter front end such that they can successfully create and verify malicious ballots. This gets hampered tremendously due to the offline usage possibility and because the voting device has no knowledge of which device will be used as a verifier. Because the verification is optional, a dishonest voter will not verify the vote. This is the case because the verifier is not needed to cast a ballot.

Process-related Analysis

Since a challenged ballot can never be cast, the verification will not reveal a meaningful receipt since an honest voter can use it for vote-selling or coercion from a ballot that is not cast. Hence, a coercer cannot receive a ballot from an actual cast ballot. This is even strengthened by the randomizer, which will blind the votes. Therefore, the process does not leak any privacy-compromising information if *honest participants* are assumed. Furthermore, the randomizer is also verified with the `Vote-Verifier-App`, which will increase security. Thus, the process becomes more secure, preventing malicious devices or entities from altering ballots.

However, if *dishonest voters* are assumed, and additional tools such as screen recorders are deployed, the voter can record the complete voting process and use this material for vote selling. Tools against screen recording can be integrated into the voting front end trying to mitigate it. Nevertheless, tools are helpless when a voter is recording the screen with an additional camera or voting with a coercer/vote buyer next to her. This is an inherent problem of any remote voting and can only be fought if voters are not allowed to vote remotely or if a voter can hand in multiple ballots while only the last one counts. A potential coercer can never be sure if the voter does not cast a ballot after providing the receipt.

The process can become safer when voters challenge the voting device with multiple verifier devices. This is the case because the possibility that numerous devices are malware-infected and work together is smaller than with only two devices. Using multiple verifier devices to verify the same ballot would allow voters to use verifiers from an untrustworthy domain. However, this also raises the extra effort a user must take to establish trust.

The extra effort a user must invest in generating trust is also criticized by [195]. It is stated that the verification removes voters' concerns about the voting system's integrity. At the time of writing, it is not apparent how voters will behave once their initial concerns have vanished. This could result in the problem that experienced users will not challenge the votes since they already trusted the system the last time they used it [195]. Stating that this behavior must be studied further in the future, it theoretically would result in less verification and, therefore, higher chances of malfunctioning devices

succeeding in casting malicious ballots [195]. Preventing this would probably only be possible in making the verification mandatory, which is not possible with the challenge-or-cast mechanism since the voting device is not allowed to know how often the verification is done.

In Estonia, where people have some experience with Internet voting, in 2013 only 3.43% and in 2014, about 4.04% of the electronically cast ballots were verified with the provided verification system [145]. Another study found that approximately between 26% and 31% of all verified ballots belonged to women, while slightly more than half of the electronically cast ballots were handed in by women [145]. Unfortunately, it is not stated why these numbers are achieved, but they could, with care, be used as an indicator for the willingness or assumed verification necessity by voters. Also, it illustrates the need to motivate all genders of voters to verify their ballots.

Technically and procedurally, it is significant that there is enough easily accessible guidance during the process, especially since the challenge-or-cast approach is rather counterintuitive. The instructions on the next step must be visible because they will be overlooked otherwise [195]. One common problem occurring is that the voters do not scan the challenge since the commitment on the voting device and the verifier looks similar after scanning it [195, 212]. Voters mistakenly believe that they already verified the ballot correctly at this point [195, 212]. Based on the process, this is not the case since the encryption has not proceeded on the verifier. Another common problem is the counter-intuitive handling since the voters are surprised that a challenged vote cannot be cast and a cast one is not challenged [212]. Here detailed descriptions or an approach where the challenged ballot can be cast would be beneficial [212].

In the challenge-or-cast approach, the verification device gets to know the selected option. This is because the verifier needs the information about the selected option to encrypt the ballot locally again. However, due to the process design, the voting device does not know the actual cast ballot's included chosen voting option. The maximum the verifier knows is the intention a voter probably has, but it cannot be sure that a voter indeed voted for the same option with which the encryption is challenged.

The code-based approach has an advantage over the challenge-or-cast approach since it verifies the connection to the voting server. This would raise the security level even more since, despite the Cast-as-Intended property, the Recorded-as-Cast feature is also verified. On the other side, the challenge-or-cast approach has advantages since no codes created by a single entity are needed.

Environment-related

The verification possibility has a significant influence on voters and their behavior. Furthermore, some points should be considered for providing and hosting the PWA. The voters can verify their

ballots to challenge the voting setup to prevent maliciously encrypted ballots. This increases the security for a voter since she must not rely on unsubstantiated trust. Hence voting becomes more secure from a voter's perspective.

Making the PWA accessible for all voters is not as trivially as assumed since if it is only provided by one entity, this entity must be a trusted one. Hence, the voters must, similar to the case with the voting authority, trust a Trusted Third Party. This trust can be avoided if various different entities host the verification application (threat *T8* and mitigation strategy *M8*). This could be achieved if the sealers, which are already distributed in a location-independent manner, furnish the application in addition to the validation of the blocks. Therefore, the need for a single TTP providing the verifier application is negligible since a user can select the sealer hosting the verifier independently. Allowing the voters to choose which provider they want to use reduces the possibility of the verifier and voting front end working together. Furthermore, it may be helpful if the voting authority provides a test environment where a voter can test the verifier before interacting with the real system. This would allow the verification of the verifier application if not provided by a TTP.

Not relying on one single randomizer entity would also enhance security. However, since the verifier application can verify the randomizer as well, the added value of multiple randomizers is not that big as the one given through multiple verifier applications.

Social engineering attacks with the objective to discourage voters from using the verifier and at the same time providing malfunctioning voting front ends can not be fought with the current setup. This is especially true because the verification is entirely optional, and there is no need to verify the voting front end at least once before voting. Hence, it is essential to mobilize voters to use the verifier application. This is classified as *Tampering* in the STRIDE scheme because it is a modification of the designed voting process.

Accountability is only partly met since if the verification fails, the user does not know directly whether the voting device or the verifier is untrustworthy. To determine which device is untrustworthy, a second verifier device needs to be used. This would allow cross-checking of the commitment generated and clearly indicate whether the voting device is malicious. If both verifiers result in the same commitment, likely, the voting device is not trustworthy. On the other side, it is an indicator of an untrustworthy verifier if the voting device and one verifier result in the same commitment.

Furthermore, voters must have a trustworthy way to report and announce malicious devices. This is currently neither implemented in the verifier nor in the voting front end. Doing so would be trivial and straightforward since it is only displaying contact information. This cannot be automated because the voting device does not know if the verification succeeded, and it is the user's decision to report

malfunctioning devices. The possibility to report malfunctioning devices would increase overall security and trustworthiness.

Comparison with other Implementations

Vote verification adds the possibility to verify the encryption solely generated on the voting device and thus allows the determining whether the voting device is honest or not. This possibility is the primary added value for a voter since it will enable voters to use a voting device they do not trust. This is definitely an advantage since trust in the voting setup can be established even in a remote environment. It allows users to verify that their vote was Cast-as-Intended which is one of the three pillars a voting system should serve.

With Provotum 2.0, as described in [148], a voter did not have the verification possibility, and there was also no Cast-as-Intended feature implemented. So basically, a voter must trust the voting client but had no evidence of whether their trust is appropriate or not. With the addition of vote verification in Provotum, this trust can now be founded on experience since the voter can determine if a voting device is trustworthy. This results in a higher overall trust level which theoretically will reduce insecurity about electronic voting systems. In Table D.2 in Appendix D this threat is listed with ID *T1* in the analysis of attack vectors and limitations. It is remedied with this extension.

Besides the newly added CaI feature, Provotum already features RaC and CaR verifiability. Furthermore, an honest voter will still not receive a valid receipt of an indeed cast ballot and cannot transfer the re-encryption proof to a third party due to using designated verifier proofs. Therefore, the RF property holds for honest voters.

In [3], the Helios system is targeted for elections with a low coercion risk. In their approach, a voter can challenge as many ballots as wished, and once the voter is convinced, she must authenticate herself and cast the ballot then. They state that all vote selections are only recorded in the browser without any network calls. Only if the ballot is encrypted and the plaintext is discarded, network calls occur. In Provotum, this is different since there are network calls to the randomizer entity after the ballot is encrypted but before the plaintext selection is discarded. This is partly caused by the nature of Provotum's encryption process and partly by storing the selection for a possible challenge of the ballot. Furthermore, the auditing of the encrypted ballot is done in a ballot encryption verification program which takes the used randomness as input. Exchanging the randomness is done by downloading it and importing it into the verifier software. Additionally, the voter gets redirected to the selection screen once she finishes challenging the encryption [3].

In Provotum or the work described by [195], the verification is air-gapped, which may not be the case with the Helios version introduced above, since there the verification program can run on

the same device. Furthermore, in Helios, the voter is bound to a computer since downloading and importing the used randomness into a specific Python program is needed. Voters will probably use the same computer they voted with, which can become problematic if this device is in the same trust domain, which may be untrustworthy.

Other systems, such as the one described in [17, 18], which are not for REV, include paper as a physical medium to transfer specific data [17, 18]. The latter uses dual voting, including electronic and paper-based voting for the same election [18]. Since these mechanisms are used onsite, they will not be discussed further here.

In the Estonian EV system, a smartphone app is used for vote verification [145, 328]. The fundamental difference in the Estonian voting system is that voters can cast multiple ballots, and only the last one is counted [289]. This brings the possibility of verifying a ballot after it is cast [145]. In their approach, an active Internet connection is needed since the verifier application needs to communicate with the vote collection. For verification, the cast ballot is downloaded to a voter's smartphone, which compares it to the encryption of all possible voting options. If there is a match between the received ballot and a newly encrypted one, the voter indicates that the ballot was Cast-as-Intended if the match occurs for the indeed selected voting option [145, 328]. A cheating malfunctioning voting device is detected if the ballot matches an undesired voting option. While with this approach, a voter gets a receipt, the nature of the voting process prevents coercion since a coercer cannot be convinced that the presented ballot was the last one cast.

Furthermore, [328] bring up the possibility of a visual comparison of the matching vote selection with the help of a color representation consisting of 32 colored cells. Whether this or the in Provotum-implemented hash icons are more straightforward for comparison depends on personal preference. The possibility to compare the values in an alphanumeric form is missing in the Estonian verification application in the form presented in [328], which may be problematic for color-blind people.

7.3 SSI-based IdM System for Provotum HE 3.0

IdM systems maintain sensitive user data and embody some of the biggest honeypots for cybercriminals. Whenever users must provide proof of identity online, an onboarding procedure is required to validate the physical certificates. After a successful onboarding process, that person's user account is considered as identified in the given context. However, the user account is loosely coupled to the physical certificate, so an early invalidation of the physical document may not be noticed immediately within the application. This can create data inconsistency between the IdM system and

the original identity context and poses a threat to systems that depend on it. An IdM system that can digitally issue credentials and verify them at all times improves this process in all those shortcomings.

In Provotum 3.0 HE, the VA is the entity in Provotum responsible for bootstrapping the system. Provotum 3.0 splits the responsibility of operating an IdM system and access control to the PBB between the VA and the Identity Provider (IdP). It is assumed that the VA runs an IdM system that maintains a list of eligible voters identified by their email addresses. At the start of a vote, the IdP receives the list from the VA and uses it to authenticate and authorize voters to submit a ballot. The goal of this separation is that the IdP cannot link registered users to the real identity of the voter.

To guarantee a higher level of security and privacy to voters, a radically different architecture for issuing, storing, and verifying credentials is proposed. The responsibility for each task is split among three independent entities. Using randomized identifiers, which are maintained on a decentralized registry and linked to cryptographically verifiable credentials, results in more flexibility and a higher level of privacy and security. These properties are enforced by design and align with Provotum's security requirements.

7.3.1 Digital Identity Management

The technological advances of the internet also introduced novel mechanisms for digital authentication. Digital identities are not limited to humans, the more generic term *subject* or *resource* is used for the person or item which is to be identified. Three primary roles exist around the issuance, management, and verification of credentials:

- **Issuers** are the source of credentials. They underline a claim about the subject in the form of a credential, like a passport issuing authority issues a document with multiple claims about the subject.
- The **holder** is responsible for storing the credentials privately and securely.
- **Verifiers** - or also referred to as the relying parties - check whether a credential is valid and grant access to the requested service.

Decentralized Identity Model (Self-Sovereign)

The goal of Self-Sovereign Identity (SSI) is to have an independent entity for each of the three roles (issuer, holder and verifier). This third iteration of digital identity is inspired by the identity system of the real world, where an identity is based on a direct relationship between the subject and its peers. The responsibility of issuing, holding, and verifying identities is no longer the duty of a single party.

Figure 7.25 illustrates how these responsibilities are distributed among the peers. This form of internet identity has recently become more prevalent under Self-Sovereign Identity (SSI).

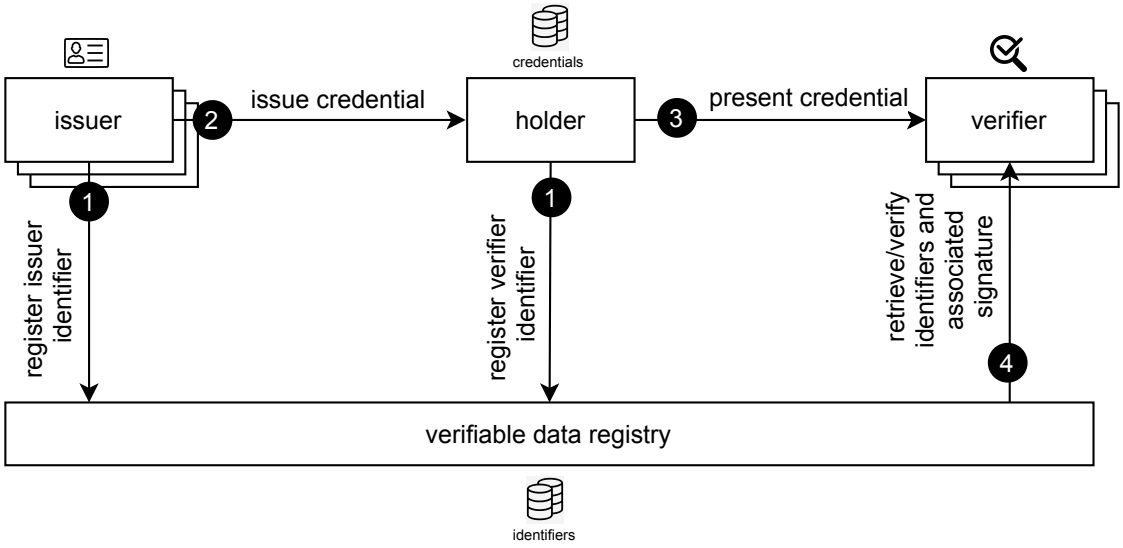


Figure 7.25: The Decentralized Identity Model Splits the Roles of Issuing, Holding and Verifying Among Three Separate Entities.

The authentication flow is best described as part of a real-world example. For this scenario, there is the government (issuer), Alice (holder) and a liquor store (verifier or relying party). Alice wants to buy a bottle of wine. The liquor store must verify that Alice is legally eligible to buy alcoholic beverages. Firstly, the government, as well as Alice, create a key pair each. From this key pair, an identifier is derived and registered publicly. The government communicates this identifier with the liquor store in a secure channel or a public credential registry (Section 7.3.6). Alice identifies herself to the government, and in return, the government issues a signed statement to Alice that contains her age. This claim is stored locally in Alice's wallet and is only known to the government and Alice. In the checkout procedure at the liquor store, Alice can present this claim to the cashier. With the help of the verifiable data registry, the liquor store can resolve the associated public keys of the identifiers that are part of the signed claim. The liquor store can verify this claim without the need to contact the government. As a result, the government does not learn when Alice uses the credential. Furthermore, with the help of a challenge-response mechanism, Alice proves that she controls the private key associated with the claim.

In addition, biometrics can be included in the claim. Alice would then have to disclose the signed image to the cashier who then can be used a face comparison by the cashier. The entire liquor store

process is carried out without the issuer noticing, since the government is not involved in the disclosure process.

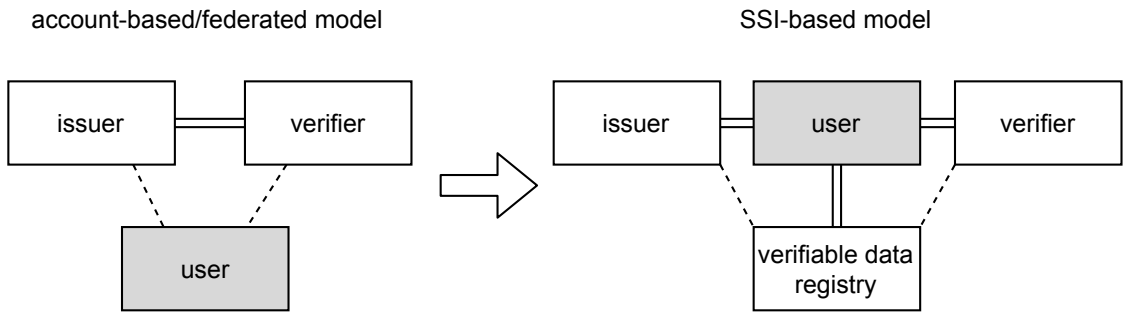


Figure 7.26: The Locus of Control that happens in the Transition from Account-Based Identity Models to the Self-Sovereign Identity Model [4]

The decentralized identity model improves the account-based and federated identity model in the following aspects:

Privacy Decentralized public key infrastructure (DPKI) enables the subject to create a direct connection to its peers. Using DLT removes the need for a TTP to verify that a claim about the subject is valid since the associated keys are publicly available. There is no need for a connection between the issuer and the verifier. Therefore, no other node in the network except the two connected peers learn about the authentication request and the data that is shared in the process. Thus, a higher degree of privacy is guaranteed in comparison to the centralized and federated identity models. Figure 7.26 illustrates how the control shifts to the edges of the network by putting the individual at the centre of the identity model. By design, the subject is in control of storing and disclosing credentials.

Data Breaches IdPs only act as credential issuers and not verifiers and therefore, they cannot track the online activity of a user. Since authentication requests are not logged by the identity issuer, the system is less vulnerable to cyber attacks that exploit large amounts of sensitive user data. To access a collection of credentials about a subject, the attacker must be able bypass the security restrictions of a voter's wallet.

Credential Management Instead of username/password combinations for authentication, DPKI is applied mitigating the risk of possible attacks such as phishing attacks [156], dictionary attacks [228], credential stuffing [197]. Private keys are maintained in a wallet. It is the wallets responsibility to enforce protection against the leakage of cryptographic material.

Portability Similar to the federated identity model, the decentralized identity model allows the subject to use the same form of identification across multiple applications. It improves the authentication process since the user must not remember which IdP was used to sign up for each application initially.

One of the challenges of a decentralized identity model is to create a user-friendly wallet recovery mechanism. Even though physical identity cards cannot be restored, people are used to a simple restoring mechanism for their digital identities. Therefore, it is desirable to have a recovery mechanism for decentralized identifiers with a similar degree of ease of use. When a user loses the private key corresponding to the digital identity and no backup was created, the identity is lost and cannot be restored. However, there are mechanisms developed that enable a recovery schema based on a group of trusted entities, often referred to as social recovery with Shamir Secret Sharing [318] (*cf.*, Section 2.5).

7.3.2 Self Sovereign Identity (SSI) Primitives

SSI is built on two pillars of standardization. Decentralized Identifiers (DID) and their cryptographic counterpart: Verifiable Credentials (VC) enable a decentralized and privacy-preserving form of digital identity.

DIDs are globally unique identifiers are at the core of SSI. SSI introduces this new type of identifier that exhibits two unique properties: *(i)* it can be created and managed without a centralized certificate authority and *(ii)* is bound to PKI such that the ownership of the identifier can be proven without a TTP. The binding problem of PKI illustrates in Section 7.3.2 the need for a new type of identifier with the mentioned properties.

Binding Problem in PKI

DIDs solve the fundamental binding problem that exists in PKI. When creating a cryptographic key pair, the private and public parts are cryptographically bound to each other. However, the controller which maintains the key pair is often non-digital. Consequently, non-digital controllers are represented with an identifier in the digital world. Usually, this identifier is not firmly bound to the key pair and thus, creating two fundamental challenges. *(i)* The introduced identifier is not strongly tied to the public key and *(ii)* the identifier is not strongly tied to the controller as illustrated in red in Figure 7.27.

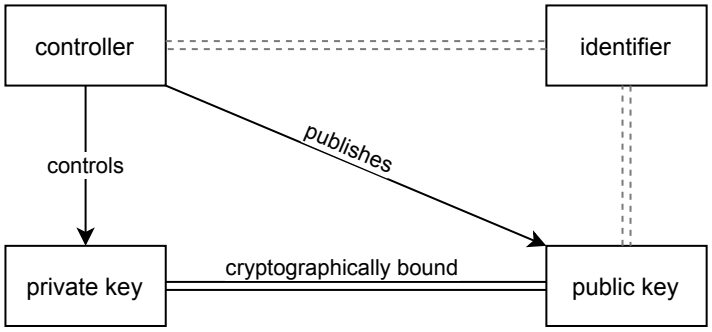


Figure 7.27: DIDs as Self-Certifying Identifiers Enable a Strong Binding Between the Public Key, Identifier and Controller

DIDs are self-certifying identifiers, meaning that the ownership can be proven without the need for a TTP. A self-certifying identifier is derived from the public key. Thus, the binding between the identifier and the public key is as strong as the binding between the public key and the private key (solving binding issue *(i)*). Furthermore, since the identifier correlates with the public key, ownership of the identifier can be proven with the private key. Therefore, only the controller can make this proof, and thus, the controller is strongly bound to the identifier (solving binding issue *(ii)*).

By binding a persistent identifier to a public key, a key rotation for that identifier is not possible. A new key pair would also result in a new identifier. However, key rotations are desired as the controller may need to adapt to newer and more secure encryption algorithms. This is solved by introducing the DID document (Section 7.3.2), which holds meta-information about the identifier itself such as the controlling public key and encryption algorithm as illustrated in Figure 7.28. Updating the associated key in this document requires the previous key to approve this change and does not modify the identifier itself. All updates together form a chain of trust. If each change is recorded publicly on a decentralized data registry, anybody can verify if the associated key controls the identifier, and therefore, the need for a TTP is removed.

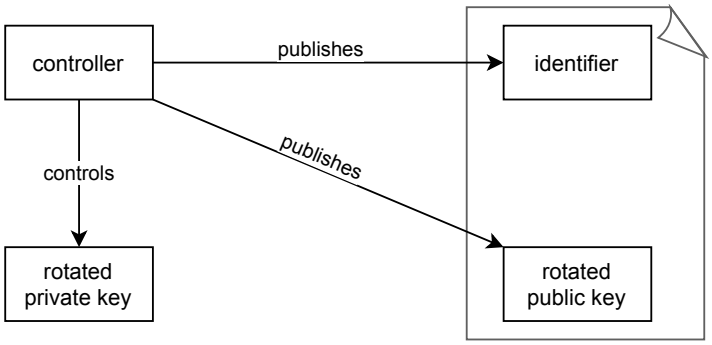


Figure 7.28: DID Documents Enable Holders to Execute a Key Rotation Without Changing the Identifier

The Scheme, DID Method and DID Method-Specific String

A DID consists of three parts separated by colons as shown in the example in 7.4: *(i)* scheme, *(ii)* DID method and *(iii)* DID method-specific string. The scheme is the first part and indicates that the URI is to be interpreted as a decentralized identifier.

DIDs are network-independent. Therefore, the DID method indicates the underlying network and makes DIDs unique across multiple networks. A standard specification is created by the W3C Working Group for implementing a new DID method [297]. At the time of writing this thesis, there are more than 70 DID method names registered in the DID Specification Registries [298]. Every DID method has its implementation of creating, reading, updating and deactivating its identifiers. These are the so-called DID operations. Their implementation depends on the underlying network. Thus, generic statements about the implementation of the four operations cannot be made. For example, DL-based DIDs usually require an on-chain transaction to update their DID document while web-based DIDs can be updated by uploading a new DID document to the storage bucket where the DID document is hosted. An overview of different types of DIDs is shown in Table 7.6. As shown in the table, DLT is not required to build an SSI infrastructure. However, the benefits of registering DIDs on a publicly DL are the data availability, immutability, security and trustless network assumptions given by the characteristics of DLT.

The third part of a DID consists of the unique identifier in the namespace of the given DID method and is called DID method-specific string. It is typically generated using random numbers and cryptographic functions. In the example 7.4, the DID method indicates that this DID must be resolved through the Ethereum Rinkeby testnet. The DID method-specific string is an Ethereum address. Table 7.6 highlights the different types of DIDs.

```
did  ethr:rinkeby  0xa8b4d9327e0776ad3765fb3baf6811e6f1390271        (7.4)
```
Scheme: did — DID Method: ethr:rinkeby — DID Method Specific String: 0xa8b4d9327e0776ad3765fb3baf6811e6f1390271

Table 7.6: Types of DIDs

TYPE	DESCRIPTION

Ledger	Maintains DIDs on the main chain of a BC or DL.
	`did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a`
	`did:sov:2wJPyULfLLnYTEFYzByfUR`
	`did:btc:12dRugNcdxK39288NjcDV4GX7rMsKCGn6B`

Layer 2	With the increased transaction costs and slow execution, layer 2 solutions on top of the base layer enable faster execution times at lower transaction costs or no fees without sacrificing security. This can be achieved by bundling multiple DID operation in batches and only store the proofs on the main chain. Crypto-economics are also required such that network operators are not incentiviced to cheat. `did:ion:EiAnKD8jfdd0MDcZUjAbRgaThBrMxPTFOxcnfJhI7Ukaw`
Peer	Identifiers that are not recorded on a globally shared registry, can be established directly within a small set of peers. `did:peer:EiAnKD8jfdd0MDcZUjAbRgaThBrMxPTFOxcnfJhI7Ukaw`
Static	Static identifiers can only be created and resolved, but not updated or deactivated. They can be algorithmically be resolved without requiring any data other than the DID itself. This is best described as a wrapper around a public key. `did:key:12dRugNcdxK39288NjcDV4GX7rMsKCGn6B`

Others	The DID specification is flexible enough such that other forms of DID methods can be created. DIDs on other networks such as git, IPFS or web exist.
	`did:git:625557b5a9cdf399205820a2a716da897e2f9657`
	`did:ipld:12D3KooWMHdrzcwpjbdrZs5GGqERAvcgqX3b5dpuPtPa9ot69yew`
	`did:web:uport.me`

DID Document

The DID document contains information on its associated subject. In theory, it is possible to attach arbitrary information about the subject to the DID document. However, this document is publicly accessible, and for privacy reasons, only the minimum necessary information should be incorporated into the document. This includes one or more public keys that are used to verify the ownership of DID during an authentication process. In addition, services associated with the subject can be attached to the document. A wide range of protocols can also be incorporated, such as an HTTP endpoint to a cloud agent that provides an interface for exchanging information directly between peers in a private and secure channel (DIDComm Messaging [140]). The data format for DID documents is JSON-LD (Section 7.3.2).

JSON-LD Data Encoding Standard

JSON-LD is a data encoding format that extends the JSON format. This standard is also maintained by the W3C consortium [288]. The intended purpose of this extension is the ability to represent digital objects and link these resources in a standardized manner. In the context of data sharing, data vocabularies or schemas are necessary tools for representing data accurately from the point of creation to sharing and verification. If every application chooses to represent data differently, it creates incoherency between the entities.

Furthermore, a JSON-LD document can be serialized in the same way as JSON documents are parsed. The JSON-LD data format specifies a number of reserved keywords that are part of the language [249]. The most relevant keywords for understanding the following sections are explained in Table 7.7.

A JSON-LD processor is a program that serializes JSON-LD documents and creates a context across multiple documents. There exist multiple techniques on how the context of a document can be communicated. The simplest form is to host the JSON-LD document on the same link that is specified in the context property.

Table 7.7: Reserved Tokens in the JSON-LD Namespace that are Relevant for this Solution

Keyword	Description
`@id`	This reserved token is used for unique identification by this property.

`@type`	Base on the properties of a document, the type can often be implied. However, in order to mitigate ambiguity the type keyword can be incorporated as `"@type": "http://schema.org/Person"`. An array of types indicate that a document is composed of multiple types. Furthermore, this token is used to represent a typed value which is a value with an associated data type.
`@context`	This is a reserved keyword that defines a common vocabulary that is used to describe the document. It maps terms to Internalized Resource Identifiers (IRIs) or nested structures. Terms are non-reserved strings in the JSON-LD namespace while IRIs identify resources. The context is used to serialize the rest of the document. `name` can be used as a shorthand notation for the following IRI. The same applies for the property `image`. Additionally, this property must be identifiable as an identifier that is an IRI. Alternatively, to the illustrated in-line context definition, which does not require a connection to the internet to serialize the document, the context can referenced by an URL which hosts the context definition. Furthermore, the context may contain a list of context to indicate that document is composed of multiple external context definitions.

DID Resolution and DID Resolvers

The process of resolving a DID document for a given DID is called DID resolution. In the context of the DID method, it represents the read operation. The DID resolver is a program that fulfils such a request. Thus, the DID resolver must be able to connect to the underlaying data registry in which the DID was created. Each new type of DID comes with a new implementation of a DID method and a DID resolver. The Universal Resolver [261] is a program that can resolve many different types of DIDs from different underlying networks. It is open source and available as a web application [69] for testing, but it can also be deployed using *docker-compose* [71].

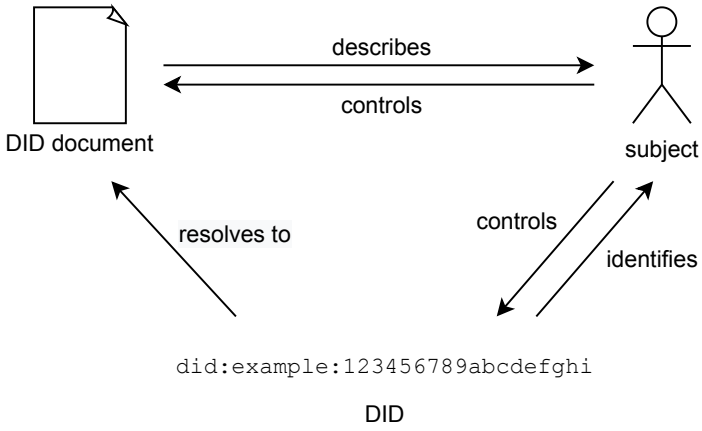


Figure 7.29: Relationship between Identifier, Metadata and the Subject

DID Auth

The objective of DID Auth is to cryptographically authenticate the subject and create a trusted connection between the holder and the verifier. This is achieved by proving control over a DID. The verifier creates a challenge in the form of a random string that is subsequently signed by the holder with the private key associated with the DID. This process can be implemented with different protocols and architectures [262]. One of the preferred methods is using JSON Web Tokens (JWT). Other possible protocols include OpenID Connect [210], TLS handshake, or HTTP.

Verifiable Credential (VC)

VCs represent the second pillar in SSI in addition to DIDs. VCs aim to create a common interface for issuers, verifiers and holders to exchange verified claims about a subject privately and securely.

A VC is used to assert personal information. It contains one or multiple attributes about a subject and is usually issued in the form of a JSON-LD document (Section 7.3.2), whereas the subject is commonly represented as DID. Other attributes of this document are the issuer, the context and type of the credential, and the signature and meta-information about the proof itself. Such claims are signed by the private key that is attached to the issuer's DID. As explained in Section 7.3.2, a DID is bound to key pair, allowing anyone to verify the authenticity of the VC. Verification only requires resolving the issuer's DID and validating the signature against the resolved public key.

Unlike DIDs, VCs are not stored in a public ledger. Instead, they are stored privately in the holder's wallet since they contain sensitive information. There are many advantages of digitally signed claims over traditional physical credentials. VCs cannot be cloned or counterfeit and can only be stolen if the attacker can access the most sensitive information which is the private key in the holder's wallet.

Furthermore, the holder of the credential has control over the disclosure of the credentials increasing privacy protection. Also, they are more portable than bulky plastic cards, and cheaper to produce.

Verifiers trust the issuer of VC and thus, together they form a so-called *trust network*. To establish a trust relationship between entities, the DID must be exchanged in a secure channel beforehand. Credential registries and governance frameworks can help communicating trusted DIDs. However, these concepts are not part of the VC or DID specification and are further discussed in Section 7.3.6.

Selective Disclosure Request (SDR)

An SDR is issued when a verifier asks the holder to reveal a set of previously issued VCs. This request is usually a JSON-LD encoded document or a JWT that contains information about the credential context and the accepted issuers along with meta-information about the request. An SDR can contain multiple credentials that need to be disclosed and the holder has to approve each of the requested claims individually in the form of a Verifiable Presentation (VP).

Verifiable Presentation (VP)

Upon receiving an SDR, the holder can either approve or discard the request. If the holder is willing to disclose the requested information, another JSON-LD encoded document or JWT is generated. It combines all the requested VCs into a single document. These VCs can originate from different issuers. This document is signed and sent to the verifier.

To only disclose minimal information, there exist two strategies: *(i)* atomic issuance *(ii)* Zero-Knowledge Proof (ZKP) enabled credentials. With atomic issuance, every VC only contains a single claim. This guarantees that a VP can be constructed from multiple atomic claims. For example, a passport has the following four properties: name, date of birth, country, gender. Applying the atomic issuance strategy results in four separate atomic VCs. When a verifier requests the disclosure of the holder's age, only one of the attributes is included in the VP which would not be possible if they were issued as one VC.

Using ZKPs enables the holder to only reveal certain fields of the VC or to prove information about a claim without revealing all the attributes of the VC. This strategy is more complex and does not work with every type of signature. Multi-message digital signature schemes (such as BBS+ signatures) are used for creating ZKP in the context of linked data such as JSON-LD. These signature schemes allow an issuer to sign an array of messages, rather than a single string. This puts the holder in control of disclosing only a subset of the values in a VC while maintaining verifiability over the VC [4, 307, 317]

Fork of the Verifiable Registry

If a decentralized verifiable data registry faces a hard fork, registered identifiers exist on both networks. This can lead to inconsistencies. The DID resolver has to decide which chain is considered as the source of truth. In the case where no consensus exists as to which fork is the mainnet, the DID resolver has to be adjusted accordingly. For example, the Ethereum community solves this issue by having a DID resolver for each chain ID. In the scenario of an Ethereum fork, the DID resolver [70] would reject any DID that solely uses the `ethr` DID method (without a network identifier) as it is not up to the resolver to decide on which chain is valid. Thus, it is recommended to use the chain identifier as part of the DID to mitigate data inconsistencies as for the Ethereum Rinkeby testnet as shown in Listing 7.5.

```
did:ethr:0x4:0xpubkey
```
(7.5)

Identity Contexts

The following definition describes the concept of cloned identities (proxies) and the meaning of the original identity context. These definitions are used throughout the thesis to reason about the data dependency of credentials among different systems.

Identity Context The original identity context is defined by the issuer of a credential. In Figure 7.30, application A can request a verification from the issuer directly. For example the original identity context of a university student is defined as all the systems that directly interact with the IdM system of the university. This can be an e-learning platform from the university itself. On the other hand, in the example where a student registers for an student discounted licence at application B, the student card is shared or the student email address is verified and in return a new user account is created within the application B. This is also called an identification or onboarding process. This new user account is loosely coupled to the original identity context meaning that a change in the original context may not be noticed in application B.

Proxy Identity A proxy identity is an account with credentials that are loosely coupled to other identity contexts. The verified claims associated with that account originate from another identity context. In Figure 7.30, the newly created user account in application B is the proxy identity.

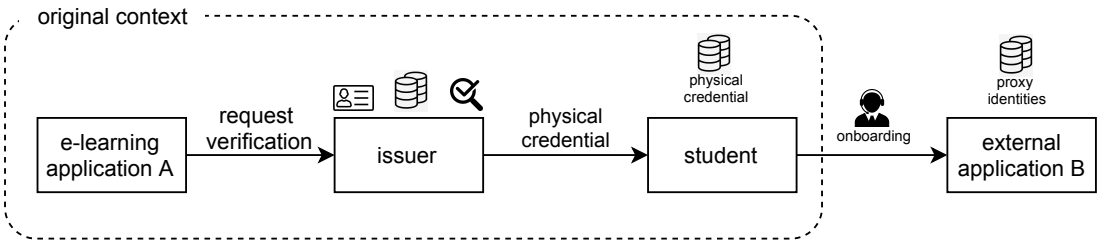


Figure 7.30: Original Identity Context Ends Whenever a Proxy Identity is Created.

7.3.3 Provotum 3.0 Security Analysis

This section performs security analysis on the eligibility verification process of Provotum 3.0 HE [148]. Only the relevant components for IdM as well as the authentication and authorization process are reviewed. In this context, the trust assumptions, threats, and vulnerabilities are evaluated in terms of security and privacy. The threat model is summarized in Table 7.8.

Eligibility Verification Process

Before being authorized to submit a ballot, a voter must go through the following sequential steps. Each step is analyzed in a separate section below and summarized in Table 7.8.

Provotum 3.0 HE assumes IdM and thus does not mention what type identifier is used to determine an eligible voter. The identifier must have a property such that the voter can prove ownership of that identifier. The prototype suggests to use email addresses as the decisive identifier. An alternative to email addresses is to use phone numbers that work in a very similar way since the ownership can also be proven with a One Time Password (OTP). But for the rest of the thesis it is assumed that email addresses are used as designed in the Provotum 3.0 prototype.

Furthermore, it is unclear how the IdP knows the correct passwords of the users. It is assumed that the VA does not share the list of passwords that are used to login to the IdM system of the VA. As a consequence, the only factor of authentication is the email address. It is assumed that a voter can use the registered email address to obtain a OTP and set a password.

(1) Registration

The VA defines and maintains a list of eligible voters, usually in the form of a centralized database. The list of qualified voters is passed to the IdP in the bootstrapping phase. Depending on the context of the identity management (IdM) system, an entry in that system often requires the user to present several claims about themselves.

In Provotum 3.0 the means of authentication is defined as a user account in the form of an email address and password. For opening such an account and becoming an eligible voter, the user must undergo an onboarding procedure. Although this is conceptually not defined in Provotum 3.0, a standard registration process at an administration office of a Swiss canton is implied. In this process, users must present valid claims about themselves and the identifying person must validate those claims.

For example, when registering at the municipal administration office of Zurich, several documents must be presented such as the passport, certificate of residence, health insurance card and rent contract for the apartment [292]. If all the documents are valid, a loosely coupled proxy identity is created. In the context of Zurich's IdM system, this proxy is considered a valid citizen until the earliest expiration date on any of the documents. The proxy is represented in the form of a username/password combination that simplifies online interactions with services within same context. Although the proxy inherits the properties from the original identity, it is fundamentally different since it is a copy that is only loosely coupled to the original identity context. This means that an update in the original context may not be noticed in the context of the clone.

The following list describes the vulnerabilities in regards to the process of registering a new identity with the IdM system of the VA:

T1 – IdM Insider Attack The paper assumes that the IdM system can maintain the set of eligible voters securely. However, these IdM systems are centralized and vulnerable to insider attacks. An employee with the necessary system authorizations can inject new entries into the system and bypass the IdP's authentication process multiple times. As long as the IdM system is not monitored properly, such an attack could go through unnoticed.

T2 – Forged ID Documents The employee at the administration office validates the documents on their security properties. Either specialized hardware is used to scan these documents and algorithms analyze the security characteristics or a trained person can identify the security features by eye. Either way, the output of the analysis of these features is probabilistic. There is no 100% guarantee that the documents are issued by the authority that they claim to be. The evaluation of physical governmental certificates is slow, expensive, error-prone and the outcome is not binary. There is a chance that forged identification documents pass the verification of a scanner or the employee evaluating the document.

T3 – Loosely Coupled Identity Contexts If one of the documents becomes invalid before the expiration date and the administration office is not notified immediately, the proxy identity remains a valid citizen inside the context of the IdM system and may still use the services as a valid citizen. This is the consequence of the chain of loosely coupled dependencies and derived

identities. Ideally, the user has to prove the validity of these documents with every meaningful interaction. However, with the current infrastructure of credential issuance, this is too cumbersome and as a result, many systems have adapted to the proxy identity pattern. Creating new identity contexts that are loosely coupled, inherits the risk of data inconsistency where the same credential is valid and expired in two different contexts.

(II) List Communication

The set of entitled voters is generated at the time of the election creation and passed to the IdP in the form of a list of unique identifiers. The following list describes the vulnerabilities in regards to the process of communicating the list of eligible voters to the IdP:

T4 – Self-Explanatory Identifier Email addresses contain linkable data such as a name, birth date, company name etc. This may enable the IdP to link a voter to the real identity and the IdP can learn which voters have registered.

T5 – Reused Identifier Anonymized email addresses can be linked to a voter's real identity if the same email address is also used for other services. It may be listed on a company's website, leaked from an online store alongside the name and address of a user, published as part of a social profile, etc. Thus, the IdP may be able to link the voter's email address to the real identity and learns which voters have registered. The content of the vote is unknown to the IdP. The risk of linkage can be minimized by creating a new email that only interacts with the Provotum system. However, it is out of Provotum's control to make sure that the email is not used for other services too.

T6 – Email Account Hijacking Any password-based authentication is vulnerable to dictionary attacks [228] and credential stuffing [197]. The enforced security rules on the voters' email accounts is outside of Provotum's control. However, the Provotum's security is only as strong as its weakest link. Using emails for the authentication process creates a security dependency on the email provider and the email client. If the email provider does not enforce strong passwords, the voter's email account is at risk and so the ballot. Such an attack would be noticed when the user attempts to authenticate with the IdP.

T7 – Static Eligible Voters The paper does not describe how this list is exchanged. Adding or removing a voter from the set of eligible voters after the election has been created is not possible.

T8 – Limited Flexibility Requiring the VA to maintain an IdM system, restricts the flexibility and complexity of authentication configurations. As stated in *T3*, incorporating new credentials into the IdM system is expensive, slow and susceptible to errors. The following example illustrates this issue. The VA wants to make an poll where only people can participate that are vaccinated against Covid19. There are two options to verify if a voter is vaccinated. *(i)* If a digital vaccination card is available from a TTP, the VA can verify the validity from this service. This raises privacy concerns since a third party is involved in the process. *(ii)* If the vaccination card is also available as a physical certificate, the VA must verify these documents in a error-prone verification process as further explained in *T2*. Both solutions are not desirable and thus, the flexibility of the election configuration is restrictive.

(iii) Authentication

The IdP acts as a TTP. Its role is to blindly sign the public key of entitled voters that can later be used to cast a vote. The following list describes the vulnerabilities in regards to the authentication process:

T9 – DoS Attack on IdP If the service of the IdP is not available, voters can no longer request blind signatures and cannot cast a valid ballot. It represents a SPOF of the system. If the service of the IdP is not available, voters can no longer request blind signatures and cannot cast valid ballots. Attacking the webservers of the IdP can make the system halt.

T10 – IdP Voting The IdP must not prove that the circulating signatures were only issued upon successful authentication. No mechanism prevents the IdP from creating new addresses, blindly signing them and casting valid ballots. Note that this would only be noticed if the IdP casts more ballots than there are eligible voters defined by the VA.

T11 – Targeted Advertising The prototype suggests using SSO as an alternative to an email address and password login. The incentives of many SSO providers is to aggregate lots of metadata about their users and monetize these detailed user profiles. Integrating a commercial SSO provider leaks information. The provider learns if the user has already registered for a certain election alongside metadata such as the location, time, web sites visited prior and after the registration, etc. Depending on the size and reach of the provider, this information can be connected and used for targeted and manipulated advertisement. [44, 264]

T12 – Email Provider Attack To prove ownership of the registered email address, the IdP sends OTPs to the voter to prove ownership of the identifier. Although this is not yet implemented, this is the assumed processes. Popular email clients such as Gmail and Outlook do not enable

end-to-end (E2E) encryption allowing them to read the contents of your emails. If an OTP is leaked, the user would not be able to authenticate with the IdP. This would be noticed. The same problem occurs with ISPs and OTPs sent in SMS. This would be less problematic if the VA communicates two authentication factors with the IdP such as a password or phone number. However, this increases the chances of threat $T4$ and $T5$ where the IdP can link a user to his real identity.

T13 – Email Provider Leak Due to the missing E2E encryption the email provider learns which voters have registered for an election.

T14 – VA and IdP Collusion In most use cases, the VA has sensitive information about its users and can identify them. Thus, Provotum 3.0 outsources the eligibility verification to an independent entity (IdP) that ideally cannot link the identifier to the real identity. However, the IdP and an insider of the VA with reading access to the IdM system can collide to find out which users have cast a ballot and which did not.

(iv) Authorization

After being successfully authenticated the voter can register on the PBB. This direct interaction with the DL has to be made before any vote can be cast. To register, the voter must submit a public key that is blindly signed by the IdP. Whenever a vote is cast, the network validates if the transaction originates from a registered public key. The following list describes the vulnerabilities in regard to the vote authorization process:

T15 – Binary Registration Provotum's PBB distinguishes between registered and not registered addresses. However, a registered voter may not be eligible for every election that is made on the Provotum PBB. As a result, it is not possible to host two elections in the same Provotum network for two different sets of eligible voters.

Table 7.8: Threat Model of Provotum 3.0 in Regards to IdM

ID	TITLE	THREAT
T1	IdM Insider Attack	VA insider can inject forged identities into the IdM system.

T2	Forged ID Documents	The process of evaluating physical identification documents is error-prone since it relies on probabilistic evaluation methods.
T3	Loosely Coupled Identity Contexts	Loosely coupled identity contexts and proxy identities result in data inconsistency.
T4	Self-Explanatory Identifier	Email addresses often contain personal information. This may allow the IdP to link the voter's email address to the real identity.
T5	Reused Identifier	Email addresses are reused among different services. This may allow the IdP to link the voter's email address to the real identity.
T6	Email Account Hijacking	Enforcing strong passwords on email accounts is out of Provotum's control.
T7	Static Eligible Voters	A voter that loses the required credentials during election runtime, can still cast a vote since the list is defined during boostrapping
T8	Limited Flexibility	Configuring complex authentication configurations is expensive, slow and error-prone.
T9	DoS Attack on IdP	A DoS attack on the IdP can make the entire system halt.
T10	IdP Voting	The IdP can generate and sign public keys and use them to submit a valid vote.
T11	Targeted Advertising	Targeted advertisement campaigns enabled by the SSO provider.
T12	Email Provider Attack	Email is the only authentication method and emails are not E2E encrypted allowing an insider to use the OTPs to register.
T13	Email Provider Leak	Email provider learns which users have requested a registration OTP.
T14	VA and IdP Collusion	Insider of VA and IdP can collide to find out which user have not registered.
T15	Binary Registration	A voter is either registered for all or no election.

7.3.4 Design of an SSI-based IdM System

This section describes the proposed architecture illustrated in Figure 7.31. All processes relevant to the correctness and the configuration of the Provotum voting system are mostly unchanged. The focus of this thesis is on the authentication and authorization process. The proposed design incorpo-

rates the concepts of an SSI-based IdM system to achieve a secure and privacy-preserving eligibility check. The new design consists of the same entities as in Provotum 3.0 HE (*cf.*, Section 7.1.1) alongside Issuing Authorities (IAs) and a public data registry which is a public permissionless BC that manages the lifecycle of decentralized identifiers (DIDs).

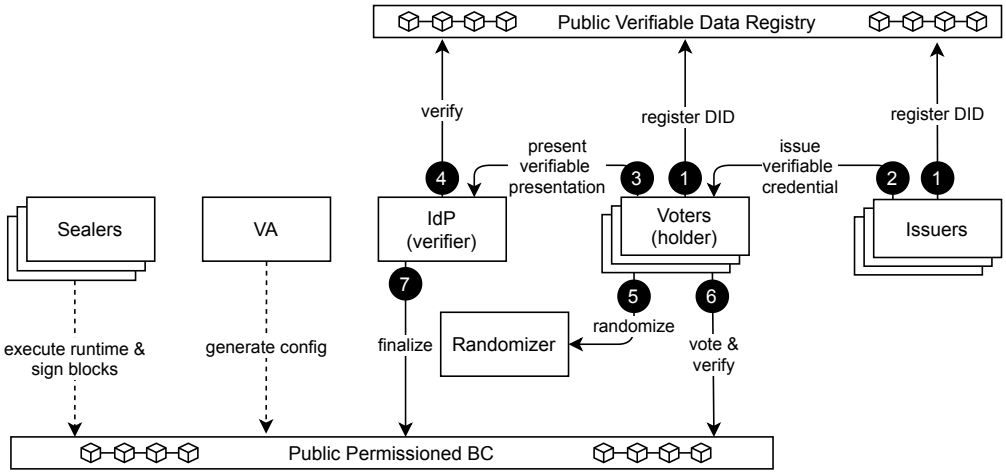


Figure 7.31: Provotum 3.0 Extended with A Decentralized IdM System

- The Substrate PoA DL acts as the system's **Public Bulletin Board PBB**. The DL implements the Provotum protocols directly into its runtime, exposing an API with which it is possible to submit transactions and read the state tied to the voting process. The PBB is also used as the communication channel for all entities.

- As with the previous iteration of Provotum, the **Voting Authority (VA)** is responsible for the bootstrapping and orchestration of the Provotum system. Coordinates distributed key generation (DKG) with sealers, creates election questions, and closes the election by signalling the sealers to start tallying. In the previous implementation, a list of eligible email addresses was passed to the IdP. This functionality is replaced with an SSI-credential-based authentication process. Newly, the VA must define the authentication configuration, specifying the claims that voters must present to the IdP. As an example, this configuration may contain a claim such as "the voter must have a valid Swiss passport and the date of birth must be more than 18 years ago".

  In the context of SSI, the classical role of the verifier is split among the IdP and the VA which is unique to Provotum. While the VA defines the claims and trusted IAs, the IdP executes and validates the received claims.

- An **Issuing Authority (IA)** is a TTPs that creates signed claims about a subject in the form of VCs. These credentials must be disclosed to the IdP during the authentication process. In SSI terminology, the IA has the role of this issuer. The VA, IdP and the IA form a trust-network. The VA has to trust the IA that credentials are issued truthfully and the IdP that claims are verified. The IA task could be carried out by the administration office, which also issues physical passports and other identification cards. Furthermore, it is also possible to invalidate the credentials by maintaining a list of revoked DIDs [1]. Revocation lists are also maintained publicly on the data registry, enabling the IdP to check the validity of a credential at runtime without contacting the IA.

- **Voters** are eligible people who participate in the election. They interact directly with the PBB when casting their vote. To do so, they are responsible for the private key management. With the integration of SSI concepts in the process, they are in charge of managing DIDs and VCs. In the SSI context, the voter has the role of the holder.

- The **Identity Provider (IdP)** receives VA instructions in the form of the authentication configuration on the task of access control to the election. Voters are asked to reveal their credentials to register on the PBB. The IdP validates those credentials with the help of the data registry. Thus, in terms of SSI, the IdP takes the role of the verifier.

  Theoretically, it would be possible to validate VCs directly in the Provotum runtime. However, the proposed design does not omit the IdP for privacy reasons. If VCs were evaluated directly on the PBB, the IAs of the VCs learn which users have cast a vote. One can argue that with the proposed architecture, the IdP gains this knowledge instead of the IA. However, the IA usually has detailed information about the user. For example, the passport issuing authority controls very sensitive information. During the Provotum authentication process with the IdP, voters may only have to disclose their nationality and date of birth on the passport to prove their nationality and adulthood. This limited information does not allow the IdP to link a voter to his real identity.

- **Sealers** secure the network by running DL validator nodes. They also participate in the DKG during the bootstrapping phase which is used for the vote encryption process.

- The **Randomizer** blinds the voter's ballot to achieve receipt-freeness. A blinding factor is used to make it impossible for a voter to reproduce the same ballot, making vote-buying much harder.

Processes

There are three new processes introduced: *(i)* credential issuance by the IA to the voter, *(ii)* the authentication configuration by the VA, and *(iii)* the disclosure and verification process between the voter and the IdP. Interactions such as bootstrapping the system or casting a vote have not changed and are not covered. These workflows continue to work as in Provotum 3.0 HE [148].

(1) Identity Issuance IAs act as TTPs in addition to IdP and randomizer. The difference between these entities is that IAs also exist outside of Provotum's context. Therefore, the registration and maintenance of DIDs must not be recorded in the Provotum runtime. Instead, a public permissionless BC is used to maintain DIDs since anyone has to be able to read the registered DIDs. The network that maintains DIDs and resembles the foundation of a digital identity should not be governed by a set of selected trustees, and the network should be as censorship-resistant as possible. Due to the available tools and libraries, Ethereum as a public permissionless BC is chosen but can be replaced with any other public permissionless BC.

Before creating a VC, the IA and the voter must first create a DID. Ethereum uses the Elliptic Curve Digital Signature Algorithm (ECDSA) to derive the public key from a private key. With the generation of a key pair, a valid DID is created accordingly (Multiple DIDs are created since this key pair can also be used on Ethereum's testnets.) without the need of an on-chain transaction.

Usually, the IA requires an onboarding procedure in which users must identify themselves independently of physical or digital credentials. If the user passes the identity check, the user's newly created DID is communicated to the IA. The IA creates one or multiple atomic VCs containing all the verified attributes, signs them, and passes them on to the voter. The voter stores the VCs locally in a wallet.

Ideally, an IA for commonly used identification documents such as passports, driver's licenses, etc., uses the same DID for all VC of the same type to its customers. Since the issuing DID on a VC substitutes all the security features of a physical passport, publicly known DIDs of such authorities enable the government to publicly maintain a curated list of trusted issuing authorities. Ideally, this list is maintained as a credential registry. On the other hand, the voter is encouraged to use a new DID for each VC that is issued. This guarantees a higher degree of privacy since a verifier is not able to link multiple VCs to the same identity when they are disclosed in independent authentication processes.

To create a common context for VCs among issuer, verifier, and holder, Schema.org is used to define the vocabulary used in the credential. Using Schema.org is not required but is a recommended practice, as it makes the credentials interoperable with other systems. New schemas can be defined if needed.

(2) Authentication Configuration Defining the set of eligible voters is fundamentally different to Provotum 3.0. The VA is not required to maintain the list of eligible voters. Instead, an authentication configuration is created and published which consists of a list of tuples. Each tuple has two properties:

- A list of required credentials must be disclosed in the authentication process with the IdP. Each claim itself contains a list of accepted IAs in the form of DIDs. Furthermore, a reason can be stated to communicate to the voter why this claim must be disclosed.
- Interpretation and evaluation of the disclosed value is not part of the SSI framework. The standard does not incorporate rules or instructions on how the value of a claim will be analyzed. Thus, the configuration also instructs the IdP on how each value of a claim must be evaluated.

As an example, the VA creates an election that only allows adult voters. The requested credential is the date of birth. The list of accepted IAs could consist of all the passport-issuing authorities in Switzerland. The included reason could state that only adults can participate in this particular election. The constraint declaration would be defined so that the disclosed value must be more than 18 years old.

Note that the two-fold configuration also applies to the physical credentials verification process. Although the government defines how to identify the validity of physical credentials, the application of access restriction rules is not within the same scope. It is up to the verifier to analyze and compare the disclosed value. One verifier may do the calculation of an age verification solely in the head, while others use a calculator.

The VA can enforce complex authentication restrictions by including multiple claims in the authentication configuration. If the set of eligible voters is too complex to define with existing VCs, the VA can also become an IA and issue custom VC to the voters themselves. However, this requires the VA to maintain an IdM system.

(3) Identity Verification Although incorporating more trusted entities in the form of IAs seems to make the system less decentralized at first glance, this is not the case since in both architectures there must be one trusted source of truthful identification issuance. This role is assigned to the VA or a trusted IA. Using a governmental IA benefits from the fact that the original identity context is used. Thus, the risk of data inconsistency between the original identity context and the internal IdM system is eliminated, and it makes the system less error-prone, more efficient, and privacy-preserving. One might argue that the introduction of IAs revives the Access Provider (AP) from Provotum 2.0 [179]. This entity acted as a gatekeeper and increased privacy by decoupling the authentication process from

the election process. The key difference between the IA and the AP is that IAs also exist outside of Provotum. SSI-based authentication enables Provotum and relying parties, in general, to directly connect to IA's IdM system without the need for cloning the identity into a new context. Original credentials signed by the trusted authority are used as a means of authentication.

User accounts do not exist in the proposed architecture. One can imagine the authentication process as if a voter had to disclose the real identification document when casting a ballot. Since a digital version of the passport can be easily verified, the overhead of disclosing and verifying the document is minimal.

The IdP reads the authentication configuration from the PBB and creates a Selective Disclosure Request (SDR). The SDR contains the required claims with its accepted IAs and a reason for each request. The SDR is signed by the IdP and sent to the voter. Since the authentication configuration is stored publicly, the voter can verify that the IdP does not ask for more credentials than the VA defines. In addition, the voter can verify that the SDR was signed by the IdP as the DID of the IdP is also recorded on the PBB.

Upon receiving the SDR, the voter has control over the disclosure of previously issued VCs and must give permission to share them with the IdP. By doing so, a Verifiable Presentation (VP) is constructed containing the signed credentials. By proving ownership across all DIDs that are used in the VCs, the VP cannot maliciously reuse these VCs since the IdP has no access to the associated private keys.

The signed VP undergoes a three-step verification process by the IdP. *(i)* The VP is validated against the SDR to make sure that the VP contains all the necessary VCs and is signed by the required IA. The IdP does not have to contact the IAs to verify that the credentials are valid. Instead, the DIDs of the IAs are resolved directly from the verifiable data registry. This ensures that the IA is not involved in the authentication process. *(ii)* The IdP checks its database if any of the VCs were previously used for the same election. *(iii)* The constraint declaration is evaluated for each disclosed VC. If all three checks are valid, the IdP stores the VCs locally and returns the blindly signed public key to the voter.

The proposed IdM system requires a radical change in the way credentials are issued and stored. By not incorporating the IA in the authentication process, they do not learn when their issued VCs are used and the voter's privacy is protected. Since atomic VCs are used where only one claim is contained in a VC, the IdP only learns the minimal necessary information about the voters.

7.3.5 Implementation

In Figure 7.32 the relevant packages for the credential issuance and SSI authentication process are illustrated.

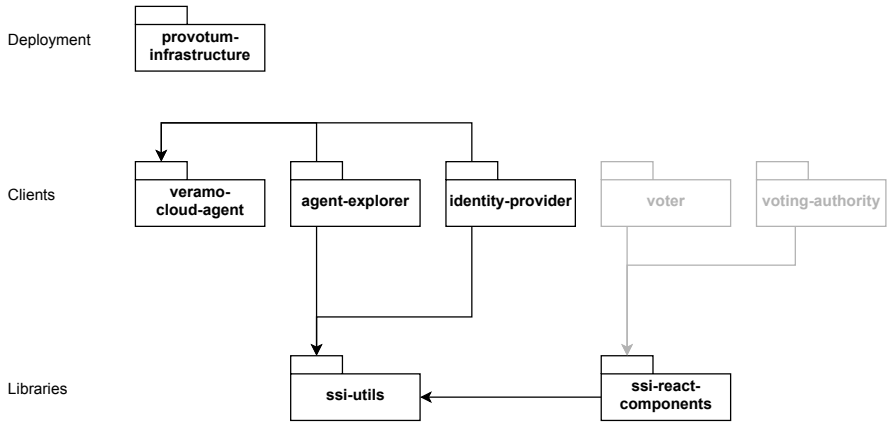


Figure 7.32: Overview of Provotum's SSI repositories

VERAMO-CLOUD-AGENT The Veramo framework [310] is used for managing all SSI-relevant processes. Veramo provides interfaces for several clients. Every entity that either issues, holds or verifies VCs runs an agent in the cloud. The service exposes all its functionality in the form of a RESTful API. Veramo currently only supports Ethereum-based and web-based DIDs. However, Veramo can be replaced with any other SSI system as long as it supports the W3C DID [250] and VC [287] specification.

AGENT-EXPLORER This implementation is a ReactJS web application that enables a GUI-based interaction with the veramo-cloud-agent. It is used by the IAs to create DIDs and VCs. Voters also use it to create DIDs, import VCs and selectively disclose VCs in the form of VPs. Some required features are missing at the time of PoC implementation, such as importing VCs, issuing VCs from predefined context templates, scanning an SDR and responding with a VP. These features are added such that an E2E process from issuing credentials to the voter to casting a vote can be demonstrated.

SSI-UTILS This TypeScript library shares common functionalities between the agent-explorer, identity provider, and the Web application of the voter and the voting authority. It includes interfaces and pre-defined schemas for VCs. It is available as a public NPM package under the name `provotum-ssi-utils`.

SSI-REACT-COMPONENTS (VOTER, VOTING-AUTHORITY) To enable modular integration and reuse of this IdM system, a ReactJS component library is created that makes it easier to integrate all the SSI functionality into new projects. Storybook[301] is integrated to show and document the usage of

the components. The library is available as a public NPM package under the name `provotum-ssi-components`.

PROVOTUM-INFRASTRUCTURE To build and deploy all the services with *docker-compose*, provotum-infrastructure was updated with the new services. All the docker images are hosted on Docker Hub.

DID Creation

As described in Section 7.3.2, DIDs are self-certifying identifiers whose ownership can be proven by their controller. The Ethereum DID method[312] using the DID registry [313] has the benefit that the creation of a DID does not require an on-chain transaction. This is possible since the DID itself contains a public key as shown in Table 7.6. If the DID resolver is not able to find any transaction associated with a particular DID on the ledger, it implies that the controller is the same public key as illustrated in the Listing 2. Thus, creating a new DID on Ethereum is fast and does not cost any network fees.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://identity.foundation/EcdsaSecp256k1RecoverySignature2020/lds-ecdsa
      -secp256k1-recovery2020-0.0.jsonld"
  ],
  "id": "did:ethr:0xb9c...e8a",
  "verificationMethod": [
    {
      "id": "did:ethr:0xb9c...e8a#controller",
      "type": "EcdsaSecp256k1RecoveryMethod2020",
      "controller": "did:ethr:0xb9c...e8a",
      "blockchainAccountId": "0xb9c...e8a@eip155:1"
    }
  ],
  "authentication": [
    "did:ethr:0xb9c...e8a#controller"
  ],
  "assertionMethod": [
    "did:ethr:0xb9c...e8a#controller"
  ]
}

```

Listing 2: Default DID Document Without Transaction History from Ethereum DID-Resolver

Credential Issuance

Issuing a VC requires a DID from the IA and the voter. The voter creates a new DID for every new credential to preserve privacy, while the issuer's DID is often required be commonly known to simplify the verification process. Section 7.3.6 discusses how publicly know DIDs can be maintained. The voter's DIDs is released to the IA. Meaningful credentials such as passports are often issued in a personal verification procedure. In this scenario, the voter's DID can be transferred offline by scanning a QR code on the spot.

Incorporating commonly used schemas such as those from Schema.org, makes VCs interoperable and more likely to be accepted by other parties. However, the underlying data structure of the JSON-LD encoding is a graph. Therefore, the schemas may contain cycles. For example, person P may be part of an organization O and organization O has a set of employees where person P is part of. Thus, programmatically extracting the fields of a given schema type is expensive as it may result in an infinite loop [136]. As a result, JSON-LD schemas as those from Schema.org are helpful for creating a common vocabulary among two entities but not for instantiating new objects for a given schema type. Creating a VC based on a given schema type, is achieved with the help of the NPM package `ssi-utils` which contains a subset of the schema types from Schema.org without cycles.

VCs are issued in the form of a JWT. Veramo uses a single message interface that processes data from other agents. Any incoming message is processed by a chain of message handlers. If any of the message handlers can successfully parse the message, it is further processed by the agent. When the IA sends the VC in the form of a JWT, the did-jwt message handler [311] parses and validates the JWT. After that, the VC can be imported into the agent.

Authentication Configuration

The VA creates a configuration that defines the set of eligible voters for a particular election. This does not require any interaction with a Veramo agent. Every election is configured with a separate authentication configuration. The configuration consists of a list of two-folded tuples. *(i)* The claim defines which credentials must be disclosed in the authentication process with the IdP. It defines the required context, accepted issuers and reason why the claim must be disclosed. *(ii)* The constraint instructs the IdP with additional restrictions on how to validate the disclosed values. This is achieved with a JavaScript expression (`js`) stored as a raw string with a placeholder (`${VALUE}`) that is replaced with the disclosed value. The JavaScript `eval()` function expects a string as input and returns the result of the evaluated expression. Using raw JavaScript strings makes it possible to define any constraint

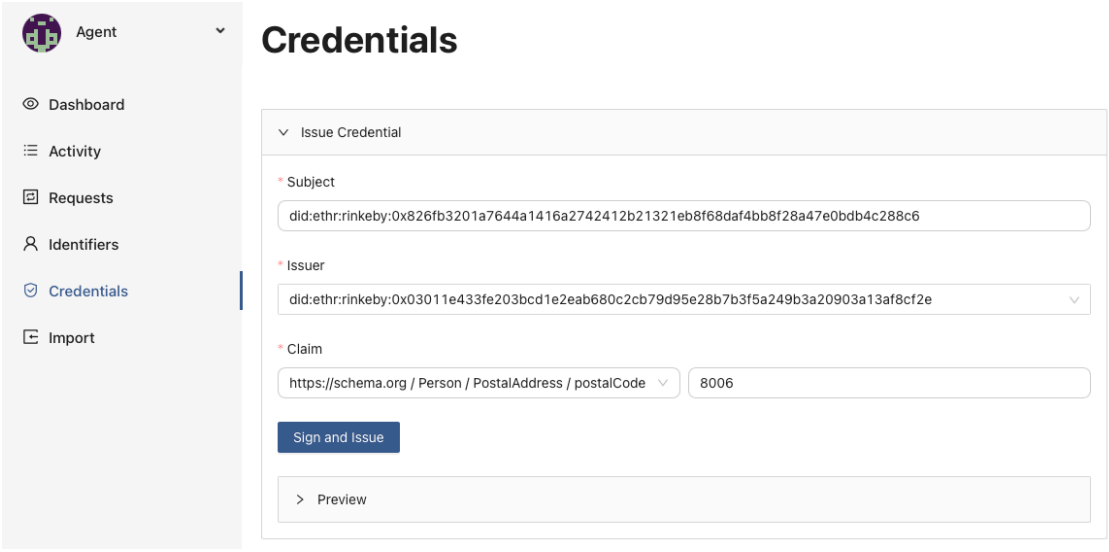


Figure 7.33: UI for issuing an atomic VC that uses Schema.org vocabulary to define a person's postal code. As a result, a JWT is produced that can be displayed as a QR Code and imported by the holder to its own agent.

that can be captured with valid JavaScript syntax. The description (`desc`) of the constraint has no functionality other than providing a human-readable string of the constraint.

Listing 3 illustrates an authentication configuration where the voter must disclose a claim that proves his postal address. This is defined by the `claim` object. Such a claim could be issued by the cantonal residents registration administration. The `constraint` adds further restrictions by only allowing certain postal codes to be accepted. To model an election without access restrictions, the authentication configuration can be omitted by submitting an empty array. This is useful for development and demo purposes.

The authentication configuration should ideally be communicated through the PBB as this does not require the VA to securely manage the configuration and also enables voters to make sure that the IdP does not ask for more credentials than necessary. Due to technical difficulties with Substrate, the configuration is currently communicated to the IdP through a separate RESTful API endpoint. The entire configuration object can be stored as a byte array since the Provotum runtime must not evaluate it and the PPB's sole purpose is for storage and communication.

```
[
  {
    "claim": {
      "credentialContext": "https://schema.org",
      "credentialType": "PostalAddress",
      "claimType": "postalCode",
      "essential": true,
      "issuers": [
        {
          "did": "did:ethr:0x030...f2e",
          "url": "https://www.stadt-zuerich.ch"
        }
      ],
      "reason": "We need to know that you live in Zurich."
    },
    "constraint": {
      "js": "['8005','8006','8007'].includes('${VALUE}')",
      "desc": "The claim must exactly match any of the elements from '['8005',
        '8006','8007']'"
    }
  }
]

```

Listing 3: Example authentication configuration requiring a voter to disclose a VC containing the postal code which is either 8005, 8006, or 8007.

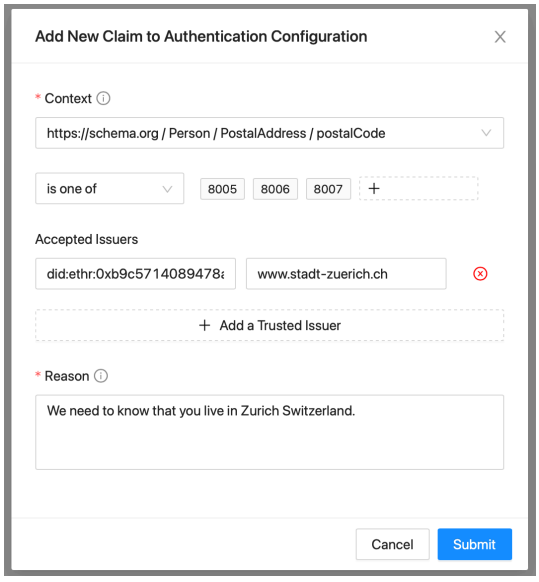


Figure 7.34: UI for creating an authentication configuration for an election. This form corresponds to the data shown in Figure 7.34.

Identity Verification

The authentication and identity verification process is illustrated in Figure 7.35. In this process, the IdP only signs the public key of a voter if the three-step identification procedure is passed. To do so, the IdP connects to its Veramo cloud agent. With the authentication configuration of the VA, an SDR in the form of a JWT is generated and signed by the agent. The JWT is sent to the voter and is displayed as a QR code in the voter frontend application. The component library (`ssi-react-components`) provides a module for integrating this logic into any ReactJS application. It has to be configured with the URL of the IdP and the election id. The component also expects the public key that is to be signed. The blindly signed public key is accessible outside of the component with a custom hook[218].

The agent-explorer supports the functionality to import an SDR by scanning a QR code. Similar to the import of VCs, the JWT is processed by Veramo's chain of message handlers. Each requested VC is listed and the voter must give permission to each claim individually as shown in Figure D.9. The voter responds with a VP in the form of a JWT. It is recommended that a DID is used for the signature of the VP that is not used in conjunction with any non-disclosed VC. This guarantees more privacy to the voter. The IdP would be able to connect previous authentication processes if the same DID was used across multiple processes. Note that this is still possible if at least one of the VCs is the same in two authentication processes. If no such DID is available, a new DID can be created. The agent-explorer is implemented such that the DID of the first VC also signs the VP to align with this recommendation.

The resulting VP in the form of a JWT is a new argument that must be sent to the IdP when requesting the blind signature. Currently, the JWT has to be copied from the agent-explorer into the voter application. The user experience could be improved using deep links and a bridge server similar to what WalletConnect[319] is providing as a communication tool between decentralized applications and wallets.

The VP and the voter's blinded public key are sent to the IdP. Veramo provides an interface for checking a VP against an SDR. It checks whether all the VCs are disclosed and issued by the correct IA. A lookup in the database makes sure that none of the disclosed VC has been used for the same election. The constraint evaluation is performed by replacing the `${VALUE}` placeholder and calling JavaScript's `eval()` function. If all three verification steps are passed successfully, the blindly signed signature is returned back to the voter. The rest of the voting process remains unchained compared to Provotum 3.0.

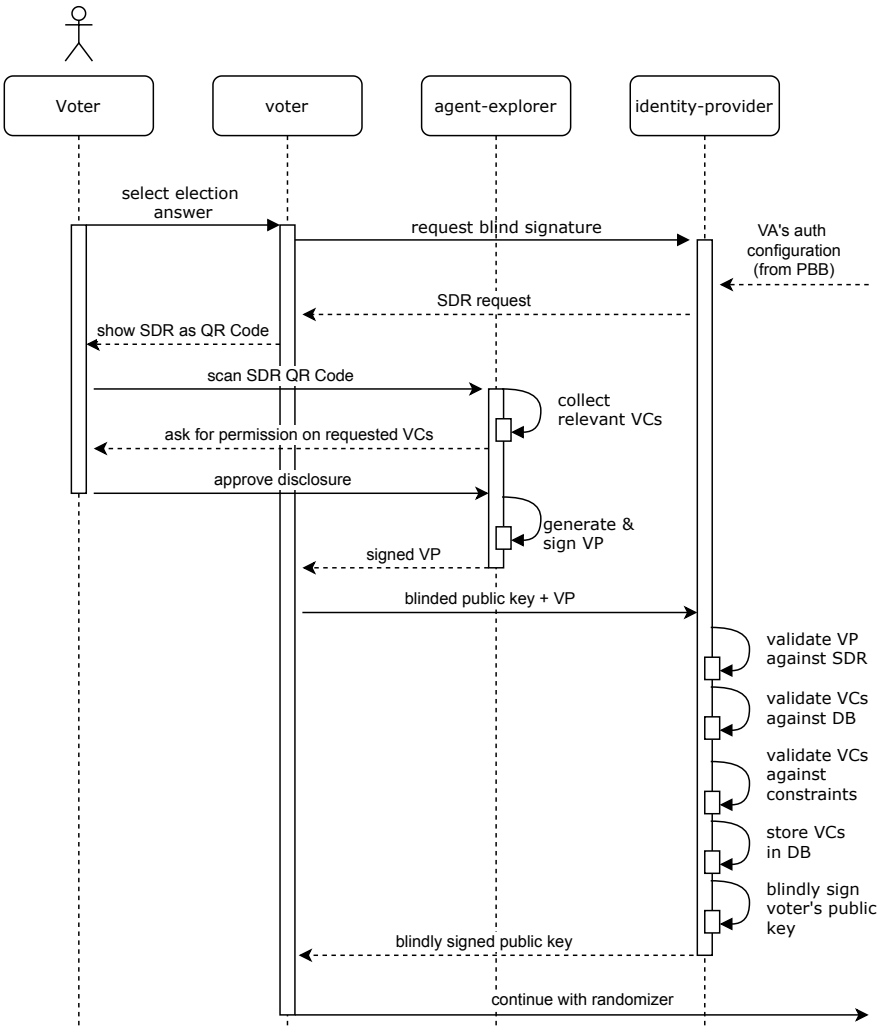


Figure 7.35: Sequence diagram for IdP authentication process

7.3.6 Evaluation

The main focus of this extension was to enable an authentication process to Provotum 3.0 HE that is secure and privacy-preserving. In this section, Table 7.9 reevaluates the identified security issues from Table 7.8 with a focus on the new architecture.

Table 7.9: Threat Model Evaluation after applying the new Design

ID	TITLE	THREAT	MITIGATION
T1	IdM Insider Attack	VA insider can inject forged identities into the IdM system.	The new architecture does not require the VA to maintain an IdM system.
T2	Forged ID Documents	The process of evaluating physical identification documents is error-prone since it relies on probabilistic evaluation methods.	VCs are native to the web. As a result, the outcome of their verification is binary.
T3	Loosely Coupled Identity Contexts	Loosely coupled identity contexts and proxy identities result in data inconsistency.	The original identity context is used.
T4	Self-Explanatory Identifier	Email addresses often contain personal information. This may allow the IdP to link the voter's email address to the real identity.	DIDs combined with VCs replace email addresses. Ledger-based DIDs usually consist of a random sequence of characters. DID documents do not hold personal information.
T5	Reused Identifier	Email addresses are reused among different services. This may allow the IdP to link the voter's email address to the real identity.	DIDs combined with VCs replace email addresses. Ledger-based DIDs usually consist of a random sequence of characters. DID documents do not hold personal information.
T6	Email Account Hijacking	Enforcing strong passwords on email accounts is out of Provotum's control.	DIDs use DPKI.

T7	Static Eligible Voters	A voter that loses the required credentials during election runtime, can still cast a vote since the list is defined during boostrapping	The authentication configuration is flexible enough to support static and dynamics sets. An invalidation of a credential is immediately available to the IdP due to the usage of the original context. Invalidating an already submitted vote is not possible.
T8	Limited Flexibility	Configuring complex authentication configurations is expensive, slow and error-prone.	Validation of credentials is done in the original context. This process is cheap, fast and simple to verify.
T9	DoS Attack on IdP	A DoS attack on the IdP can make the entire system halt.	No mitigation implemented.
T10	IdP Voting	The IdP can generate and sign public keys and use them to submit a valid vote.	No mitigation implemented.
T11	Targeted Advertising	Targeted advertisement campaigns enabled by the SSO provider.	No third party involved in authentication process.
T12	Email Provider Attack	Email is the only authentication method and emails are not E2E encrypted allowing an insider to use the OTPs to register.	DIDs in combination with VCs replace email addresses and thus, no email provider or TTP is involved.
T13	Email Provider Leak	Email provider learns which users have requested a registration OTP.	DIDs in combination with VCs replace email addresses and thus, no email provider or TTP is involved.
T14	VA and IdP Collusion	Insider of VA and IdP can collide to find out which user have not registered.	VA does not maintain an IdM system.
T15	Binary Registration	A voter is either registered for all or no election.	IdP evaluates VCs per election. (Functionality on PBB not yet implemented)

Privacy

With the introduction of a decentralized IdM design, the credential-disclosure-based authentication process is facilitated only between the IdP and the voter without the need of the IA or any other TTP. This direct communication guarantees a higher level of privacy compared to other standards such as

SSO or OIDC (*T13*). As a result of the improved privacy protection and the removal of TTPs from the authentication process, meta information about the voting behaviour is retained. Thus, voters are less likely to become targets of manipulating advertisements since less data is available to advertisement platforms (*T11*).

The replacement of email addresses with DIDs as identifiers also improves the possible threat of identity linkage (*T4*,*T5*). DIDs are randomized strings that do not reveal personal information. Also the creation of a new DID is fast and does not cost any network fees, enabling the creation of a new identifier for every application.

Identity linkage is still possible with the proposed design. It is the VA's responsibility to create an authentication configuration that is generic enough such that the IdP is not able to link the disclosed credentials to a real identity. The VA should always only incorporate the minimal number of claims necessary to define the eligible set of voters. Incorporating needless credentials into the authentication configuration increases the chance that the IdP is able to uniquely identify a voter and thus, learns which voters have registered. To minimize the risk of identity linkage, the implementation supports atomic credentials, where only one claim is issued per VC. This enables voters to only disclose the credentials which are requested. However, even with a minimized authentication configuration, the IdP may still be able to link voters to their identity through the disclosed credentials. For example, requesting a voter's postal code and date of birth may already uniquely identify a person from a sparsely populated area. Ideally, the voter only discloses a proof that states that he lives in one of the accepted postal codes and that his date of birth was more than 18 years ago. Thus, ZKPs can further improve privacy. The necessary infrastructure and the appropriate type of ZKP has to be further investigated.

IdP IA Collusion

The IdP must protocol which credentials were used for every election to prevent double voting. If the logged credentials leak, the IA can connect the credentials to additional information within the IA's IdM system and identify which voters have registered.

The Provotum architecture is not limited to national elections. For example, a student association at the University of Zurich creates a Provotum election where voters must prove their eligibility by disclosing a valid enrollment receipt. The University of Zurich acts as a trusted IA and issues such enrollment receipts to its students in the form of VCs. The IdP will only know that a voter is in possession of such a receipt but cannot link it to any personal information. If the authentication logs of the IdP leak, the University of Zurich would be able to reconstruct which students have registered since it stores personal data about its students and knows which VC belongs to which user. However,

the university does not know for sure if the registered voters have effectively cast a vote and also, their actual vote would remain private.

This challenge is introduced with the new architecture. One can argue that it replaces Threat *T14* where the same collusion was previously possible between an insider of the VA and the IdP. However, an authentication configuration containing credentials from multiple IAs increases the risk of such an attack. If any of the IAs is corrupt and collides with the IdP the voter's identity can be linked. The higher the number of IAs incorporated in the authentication configuration, the higher is the chance of an insider attack between the IdP and one of the IAs. However, the only information that can be extracted from such an attack contains which voters have registered and which have not and not the content of the actual vote itself.

Security

The introduced decentralized IdM system proposes that the necessary credentials are issued outside of Provotum. The new Provotum system uses the original identity context of each credential, eliminating the risk of data inconsistency (*T3*). As a result, the VA must no longer maintain a database with proxy identities.

The previously used centralized IdM system enabled an adversary with the necessary credentials to inject new identities and bypass the authentication procedure multiple times. The removal of the internal IdM system mitigates the risk of insider attacks. In the new design, such an attack is still possible for an insider at the IdM system of the IA. However, the authentication configuration can be constructed such that the claims of multiple IAs is required. Therefore, an attacker would need to be able to create a forged credential in the IdM systems of all the IAs making such an attack more difficult to execute.

Email accounts are subject to a wide range of attacks (*T6*). The replacement of email addresses with DIDs as identifiers and their cryptographic properties eliminates the risk of credential stuffing and dictionary attacks. Passwords are replaced with cryptographic signatures guaranteeing a higher degree of security by design. Thus, the authentication process with DIDs and VCs is based on the same cryptographic principles which are used in the vote casting process.

In the previous architecture, OTPs were used to prove ownership of the identifier. The missing E2E encryption in email communication and the use of a single factor of authentication enables an insider of the email provider or the email client to leak OTPs (*T12*). The introduction of DIDs and the direct interaction between the holder and the verifier eliminates the threat of such attacks. Proving ownership of the identifier is facilitated with cryptographic signatures without the need of a TTP. As

a result of the removal of email and SSO/OIDC providers, fewer trust assumptions have to be made in the E2E process.

Proxy identities are created after a successful onboarding process in which credentials are verified and allow users to act as identified users in a given context without having to reveal physical credentials every time. However, the design pattern of proxy identities results in loosely coupled identity contexts and data inconsistency between the original and the derived context. The new architecture which is always operating in the original identity context benefits from guaranteed data consistency that previously was susceptible due to the use of proxy identities ($T3$).

Whenever a physical credential is verified and a new proxy identity is created, the system becomes less efficient and error-prone since the validation of physical certificates relies on a probabilistic evaluation of the security features ($T2$). The proposed architecture issues credentials that are native to the digital world and always operates in the original identity context of a credential. The validation of a VC evaluates cryptographic signatures which is cheap, fast and results in a binary outcome.

Previously, every externally issued credential had to be verified by the VA and credited to the proxy identity. This verification process is expensive and slow as it often relies on human interaction. With the digitally native credentials and the usage of the original identity context, complex authentication configuration can be constructed ($T8$). Credentials of different authorities can be aggregated into an authentication configuration without any additional verification process needed.

The issuance of credentials follows an interoperable standard proposed by the W3C [250, 287]. In combination with the use of a common data vocabulary of Schema.org, the identifiers and credentials used in the proposed architecture and the prototype are compatible with the credentials used in other SSI systems. However, the disclosure and verification process of credentials is currently defined by the Veramo framework. This process does not fully align with the W3C standard, making a disclosure procedure not compatible with other SSI applications. As this is a know difference to the Veramo project, it will be addressed in a future release [277].

Infrastructure

Some components used in the architecture are heavily influenced by other services, project or standards. This section evaluates the technologies used for those components.

Veramo Veramo was open-sourced in December 2020 as a public beta version. Although it covers many SSI functionalities, some crucial features are still in development. For example, a VP may contain multiple VC which are issued for different DIDs. During the disclosure, the verifier must make sure that the holder is in control of the DID in each VC. Currently, this controller check is in research

and development and not yet integrated into the prototype. Without this check, a holder can disclose a VC that is not under his control.

In the context of VPs, another crucial functionality is still missing. When a holder responds to an SDR with a VP, the VP does not contain information stating that it is generated for that particular SDR. This is not a problem as long as the holder proves ownership for all DIDs within the VCs. However, Veramo's interface for SDRs contains a significant field in the form of the tag which allows the IdP to incorporate an arbitrary string. This becomes relevant in an architecture where the IdP must prove to the PBB that it possesses a certain number of valid VCs that have been disclosed in the context of a certain election. The tag on the SDR can be used to link the SDR and VP to an election by setting the tag to the election identifier. With such a functionality in place, every VP is tagged with the election identifier.

Centralization Risk of DID Methods At the time of writing this thesis, there are more than 70 DID method names registered in the DID Specification Registries [298]. Running a resolver for each DID method requires a connection to the underlying data registry. This task requires a large storage capacity, fast network connection and computing power that is fast enough to update the state for all the underlying networks. There are approaches to streamline the setup of a Universal Resolver [261]. However, if the number of DID methods expands further, an IdP needs to rely on third-party services to resolve DIDs. This leads to an undesired trend for centralization. A possible solution to that problem is to restrict the allowed DIDs on the issuer level and maintain a list of supported DID methods. For example, a passport issuing authority could only issue VCs to people that use a particular DID method. The list of supported DID methods could be incorporated in the DID document of the issuing authority or the governance framework (Section 7.3.6). This list is publicly available and the IdP would only need to host DID resolvers for those networks.

Incomplete W3C Specification Although the DID specification [250] is currently still declared as a working draft, some crucial components are missing. As suggested in *Self Sovereign Identity* (Chapter 11.3.3) [4], a public credential registry allows issuers to publicly register VCs to enable a wider application of DIDs and VCs. The entity maintaining such a list of public VCs is called the governance authority and publishes a governance framework. It defines the rules and policies on how an issuer can become part of that registry. Figure 7.37 illustrates how credential registries fit into the SSI architecture. The same VC that is issued to the holder, is also publicly registered such that it can be searched, discovered and verified. Such credentials are not meant for sensitive data like passports. However, they can be used for registering businesses alongside their licenses and certificates. In the

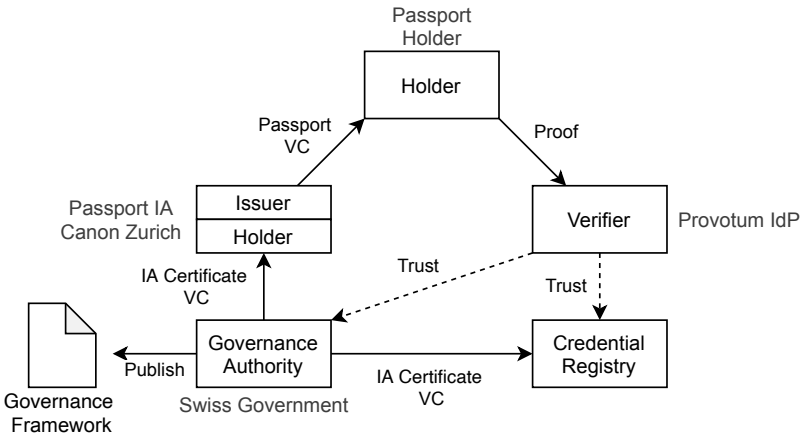


Figure 7.36: A governance framework defines the rules and policies to become a verified issuer for a given context. The example shows that the Swiss government could publicly maintain a list of verified passport IAs in the form of a credential registry. Thus, the list of eligible IAs can be discovered and verified by only knowing the DID of the Swiss government.

Provotum context, the Swiss government can publicly maintain the list of certified passport issuing authorities in the form of a credential registry. During the creation of the election, using a credential registry instead of a manually registered list of DIDs of all the trusted IAs is less error-prone. Figure 7.36 illustrates how the governance authority makes sure that only certified issuers are part of the credential registry. With such an architecture in place, Provotum's IdP could directly get the DIDs of all certified passport issuing authorities from the credential registry.

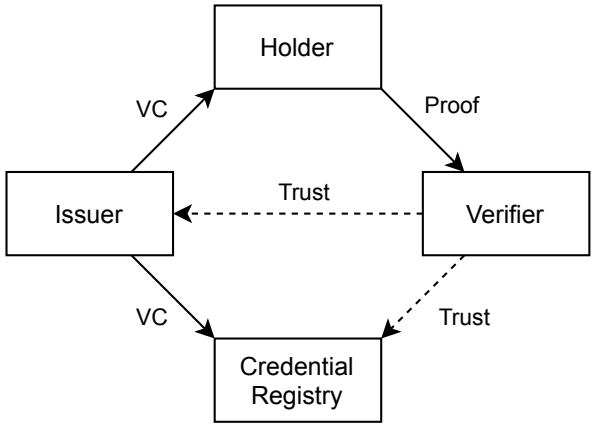


Figure 7.37: Credential registry for public VCs

7.4 ProvotuMn 3.0

ProvotuMn 3.0 is a decentralized and Receipt-Free (RF) voting system based on an end-to-end verifiable Re-Encryption Mixnet (RMN). RMNs allow for flexible votes and elections and decouple the ballot structure from the cryptographic voting protocol. RMNs use cryptographic shuffles and Non-Interactive Zero-Knowledge Proofs (NIZKP) to assure privacy through decentralized mixing among the DL consensus nodes. Thus, ProvotuMn decentralizes trust *(i)* through the use of cryptographic shuffles and Non-Interactive Zero-Knowledge Proofs (NIZKP) in an RMN executed among DL nodes, *(ii)* by employing a distributed key generation for election keys, and *(iii)* by offering a decentralized re-encryption service assuring RF [172].

7.4.1 Building Blocks

Voting schemes combine various cryptographic protocols to achieve privacy and verifiability requirements (*cf.*, Figure 7.38). A complete background on major cryptographic building blocks is given in [91, 172], as well as [130]. Essential cryptographic primitives are outlined hereafter.

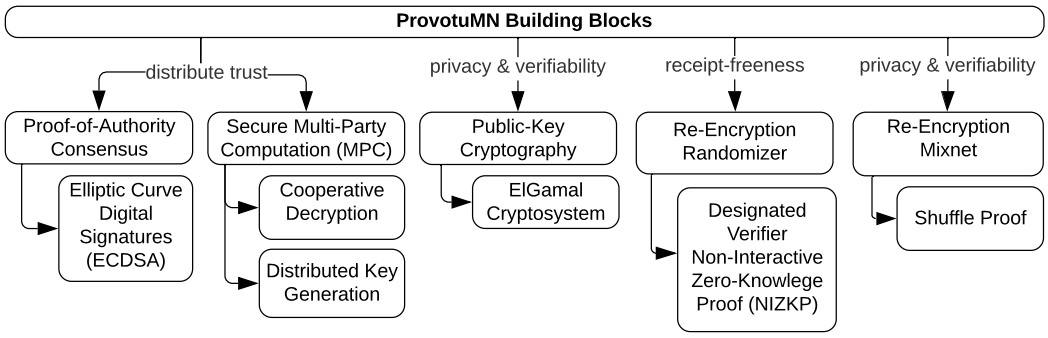


Figure 7.38: ProvotuMn's Voting Cryptographic Building Blocks

Proof-of-Authority (PoA) Consensus ProvotuMn is based on a public permissioned DL based on a fixed set of authorities operating a PoA consensus; reading is possible for anyone, new blocks are alternately produced in fixed time intervals, and the consensus is reached by a majority vote among the authorities [7]. ProvotuMn focuses on a setting in which a set of known, semi-trusted authorities exist (*e.g.*, municipalities, cantons, government), and the choice of a public-permissioned DL is evident.

Secure Multi-Party Computation (MPC) Secure MPC allows the removal of any TTP in a distributed computation. One key aspect of secure MPC in ProvotuMn takes place during the Pre-Voting Phase: the Distributed Key Generation (DKG). Based on DKG, trust can be distributed, and the risk of one party decrypting individual ballots is mitigated. Further, DKG can be used with threshold cryptosystems, where a threshold of n out of k total number of trustees is defined. These trustees collaborate in the public key generation (*i.e.*, the election key). The same keys are used to perform mixing operations in the RMN. To decrypt a ciphertext in the end, collaborative decryption must be performed by a minimum of n participating trustees, and the ciphertext decrypts to plaintext again [176].

ElGamal In ProvotuMn the ElGamal public-key cryptosystem (*cf.*, Section 2.3.1) is a fundamental building block, and it bases its security on the intractability assumption of the discrete logarithm problem in the cyclic group *i.e.*, the Decisional Diffie-Hellman (DDH) assumption. ElGamal achieves *semantic security i.e.*, ciphertext is randomized and non-deterministic while decryption remains deterministic (IND-CPA) [8]. Message Encoding is crucial when using ElGamal [94]. One method is to ensure that messages are restricted to quadratic residues in a safe prime group of order q. Thus, such encoding demands the equality check: $m^q \mod p \quad 1$ to hold, *before* a message is encrypted. If the equality does not hold, the message must not be encrypted [94].

Re-Encryption Mixnets (RMN) Re-encryption MixNets (RMN) (*cf.*, Section 3.4.4) were first introduced by [239] based on a probabilistic public-key encryption system, such as ElGamal [95] (*cf.*, Section 2.3.1), allowing a re-encryption of ciphertext. The process of a re-encryption mixnet can be compared with the physical task of shaking a ballot box to shuffle the ballots and, thus, making it impossible to associate a ballot with a specific voter [126, 130]. Formally, an RMN is based on a series of cryptographic shuffles, each composed of a secret permutation and a re-encryption function [126]. The cryptographic shuffle re-encrypts and permutes a group of ballots, *i.e.*, changes the ciphertext appearance and outputs the result in a randomized order. To avoid recoverability, this step can be repeated multiple times [131]. The RMN algorithms used in ProvotuMn are based on the CHVote specification [126], as well as pseudocode provided in [130]. The shuffle proof used in ProvotuMn was first introduced by [106] and is based on permutation matrices. The implementation is the proof of a correct shuffle [142, 304] and adopted from [126].

7.4.2 System Design

ProvotuMN 3.0 improves on the disadvantages of its predecessor systems (Provotum 2.0 [179] and Provotum 1.0 [214]) by using a verifiable RMN instead of HE, and thus, allowing for an arbitrary ballot complexity. Further, the key size is configurable, which allows for the adjustment of the security level by adapting key sizes. The stakeholders of ProvotuMN consist of (*cf.*, Fig. 7.39) a Voting Authority (VA) bootstrapping the DL, managing the vote or election; n Sealers that store the DL and form consensus in the DL network; n Re-Encryption Randomizers; and m voters. ProvotuMN design relies on k n sealers being present during both the key generation and the cooperative decryption. It is important to note that the Re-Encryption Mixnet code is embedded within the DL, and thus, it also executes in the same environment (*cf.*, Fig. 7.39).

ProvotuMn's implementation is based on Substrate [237], a modular framework that allows to create custom DLs. The ProvotuMn voting protocol and RMN implementation are contained in its own Substrate package, called *pallet-mixnet*. The source code is written in Rust and is available in [90], while deployment scripts are available in [295].

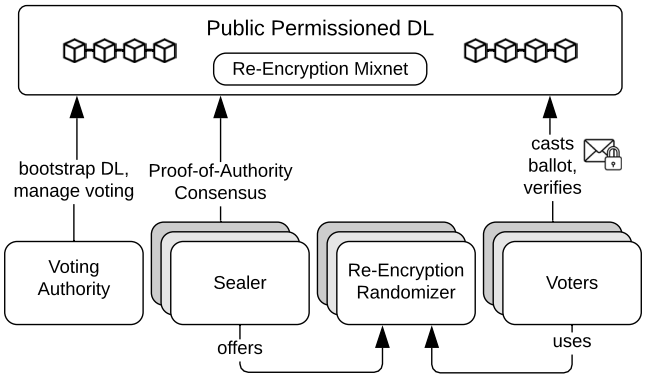


Figure 7.39: ProvotuMn Stakeholders

Assumptions

Table 7.10 shows those assumptions congruent with [175] and [179], since ProvotuMn is an evolution of those systems: a REV system that takes the requirements of federal Switzerland into account.

Table 7.10: ProvotuMn's Assumptions

ID	Description
$A1$	Voters are allowed to cast a single vote once.
$A2$	Eligible voters are uniquely identifiable.
$A3$	Voting authority nodes are fixed and known *a priori*.
$A4$	The connection between the voter's voting device and the voting network is confidential.

7.4.3 Voting Scheme

The initial pre-voting phase of ProvotuMn's voting scheme (*cf.*, Figure 7.40) is based on [179], but instead of using homomorphic tallying, ProvotuMn is based on a Mixnet, *i.e.*, sealers performing a sequence of cryptographic shuffles [126]. It consists of three phases divided into multiple protocols, as detailed in the following section. The following scheme refers to the decentralized , which is synonymous with the run-time environment of the DL, which is agreed on by the participating Sealers, which execute instructions in a decentralized fashion, adhering to the PoA consensus.

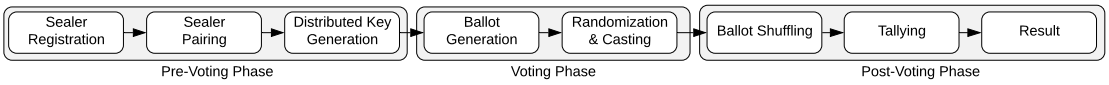


Figure 7.40: ProvotuMn's Voting Scheme

Pre-Voting Phase

During the pre-voting phase, the sealers generate DL credentials and register with the VA, which then bootstraps the DL and jointly establishes the PBB. Finally, the Distributed Key Generation (DKG) protocol is performed in a decentralized fashion. The key pair can additionally be signed by certificates (*e.g.*, X.509 certificates owned by the Sealers) to authenticate the newly generated key pair. Then, the Sealer's $^{s}_{i}$ is transmitted to the VA via an authenticated channel to ensure the process is not prone to a *Man-in-the-Middle* (MitM) attacks. Depending on the specific application, one or more key pairs are required to allow to differentiate between block production and block finalization.

2. Sealer Pairing Next, the VA generates the DL's initial configuration file (*i.e.*, genesis block), containing the sealer's 's, which authorizes them to participate in the Proof-of-Authority (PoA) consensus mechanism. The VA does *not* participate in the consensus, acting primarily as an administrative authority. Finally, the VA publishes the cryptographically signed configuration file (*i.e.*, the

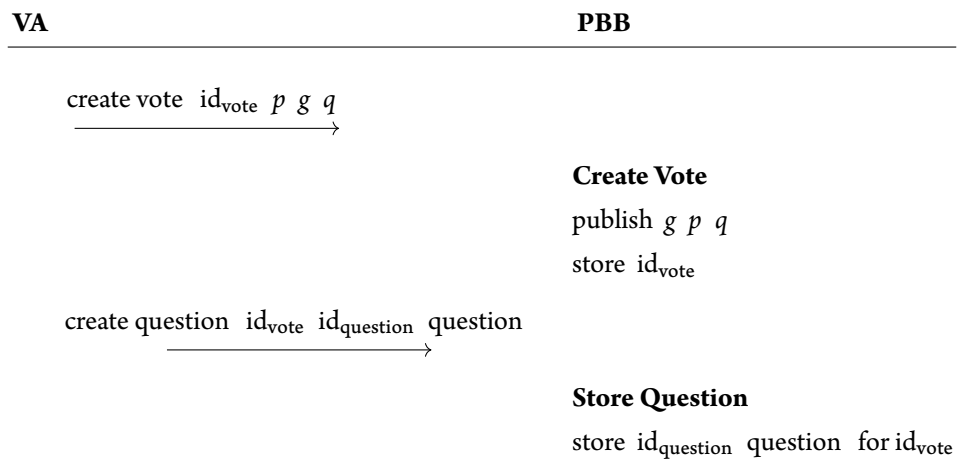


Figure 7.41: Pre-Voting Phase - Registration Step - Create Vote & Store Questions

genesis block) and starts a DL node, acting as a bootnode, allowing the other nodes to connect for bootstrapping of the DL network. In turn, each sealer retrieves the configuration via a public Application Programming Interface (API) and starts a DL node, establishing the network via the bootnode.

3. *Distributed Key Generation* The final protocol of the pre-voting phase consists of multiple sub-steps. First, the VA generates the public parameters needed for the ElGamal cryptosystem (p g q) and publishes them in a vote creation transaction to the , where the parameters are validated and included. Then, the VA sends questions that are associated questions to the vote (*cf.*, Fig. 7.41). Next, each sealer i N generates an ElGamal key pair ($^{e}_{i}$ $^{e}_{i}$ together with the accompanying Schnorr , proving knowledge of $^{e}_{i}$. The and $^{e}_{i}$ are sent to the , where the proof is validated, and then, the public key is included.

Finally, once all sealers have submitted their public key, the VA triggers the voting system's key generation (pk^{e}). The VA combines all $^{e}_{i}$ and generates the system's public key pk^{e}, which is the key used by the voters to encrypt ballots. After that, the vote is opened by changing the voting state to the next phase.

Voting Phase

The voting phase mainly concerns voters, who first select their choices, generate their ballots, then re-encrypt them using the *Randomizer*, and only then cast them to the . The phase is split into two steps: (1) Ballot Generation and (2) Randomization & Casting which is visualized in Figure 7.43.

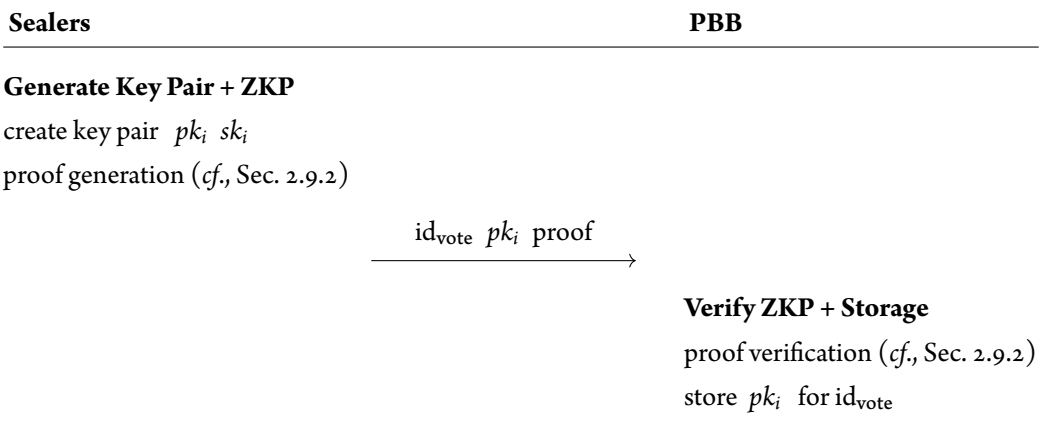


Figure 7.42: Pre-Voting Phase - Key Pair, and Proof Generation & Verification, Storage of Public Key Share

4. Ballot Generation Voters select their choices and encrypt the ballot b to a ciphertext b_v enc v r pk^e , using a random value r, as well as the vote's public key.

5. Randomization & Casting To achieve Receipt-Freeness (RF), voters interact with the randomizer and send their ballots to the randomizer, where each ballot is re-encrypted b_v re-enc b_v and signed with the randomizer's private key, as well as a , proving that the re-encryption contains the same vote v. Voters, in turn, verify the proof and submit the re-encrypted b_v to the . Before including the submitted ballots on the *(i)* first, the checks whether the b_v resembles any already included ballots to mitigate replay attacks. Second *(ii)*, the signature of the randomizer is verified to ensure that the ballot is indeed re-encrypted and thus, RF is achieved.

Post-Voting Phase

Once the voting phase ended, the VA closes the vote, advancing the system's state to the post-voting phase. From now on, no new ballots are included in the . During the post-voting phase, the ballots are cryptographically shuffled in a fully decentralized manner, followed by a collaborative decryption to plaintexts and finally, the tallying of the result.

6. Ballot Shuffling This phase is fundamental for the privacy of ProvotuMn, since the ballots are cryptographically shuffled. First, the sealers retrieve a set of all ballots $\mathcal{B}$, then perform a cryptographic shuffle and generate a Shuffle , (*cf.*, Fig. B.1 in Appendix B), rendering a new set $\mathcal{B}_{s_1}$. The computation of the shuffle is performed outside the DL context *i.e.*, in an off-chain worker, because it involves random values that need to be kept confidential for each mix *i.e.*, for each shuffle.

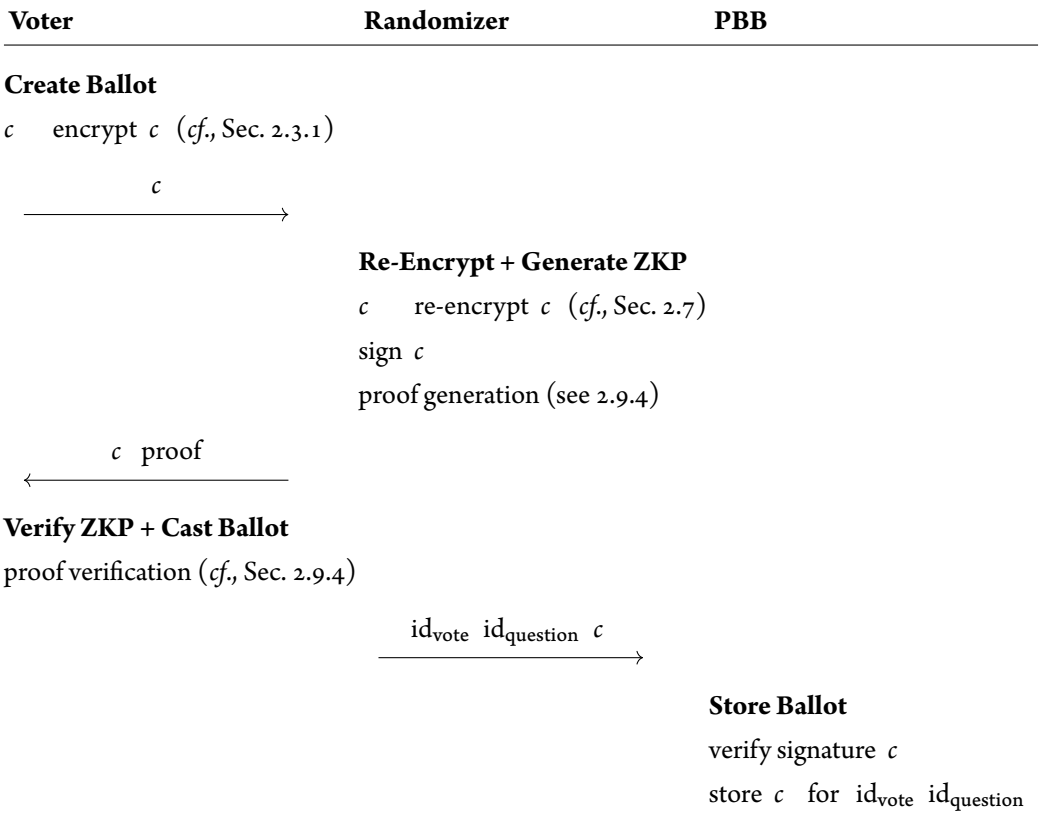


Figure 7.43: Voting Phase - Ballot Generation, Ballot Randomization & Casting Steps

Although ProvotuMn performs shuffling on the same machine as the DL consensus node is running, it can be performed in an air-gapped environment (or within a Trusted Execution Environment).

Shuffling is continuously repeated by all N Sealers in a round-robin fashion, following the same rules as the PoA consensus of the DL network. The process is repeated until the desired threshold of shuffle iterations is reached, which is specified in the initial configuration and can be adapted with each voting. It is suggested to perform more than a single shuffle to avoid recoverability in the case of a single malicious Sealer. In case of numerous ballots, these ballots can be split into batches such that multiple sealers can simultaneously shuffle a different batch of the same generation of ballots. The shuffling stops once the desired number of shuffles is reached. This number can be specified in the initial configuration and varied on a vote to vote basis.

7. Tallying Once the ballot shuffling concludes, the VA can initiate the distributed decryption of the ballots *cf.*, Fig. 7.44. For this, each sealer i N partially decrypts the shuffled ballots c . Similar to the re-encryption, the decryption also occurs outside the DL context (*i.e.*, in an off-chain worker) as it involves the sealers' private key shares $^{e}_{i}$. Besides the decryption, sealers also generate a

proving knowledge of the private key share $^{e}_{i}$, and send both the partial decryption and the proof to the DPBB. All N sealers participating in the key generation process are also required to participate in the distributed tallying. Otherwise, ballots cannot be decrypted if any sealer refrains from submitting its partial decryption key.

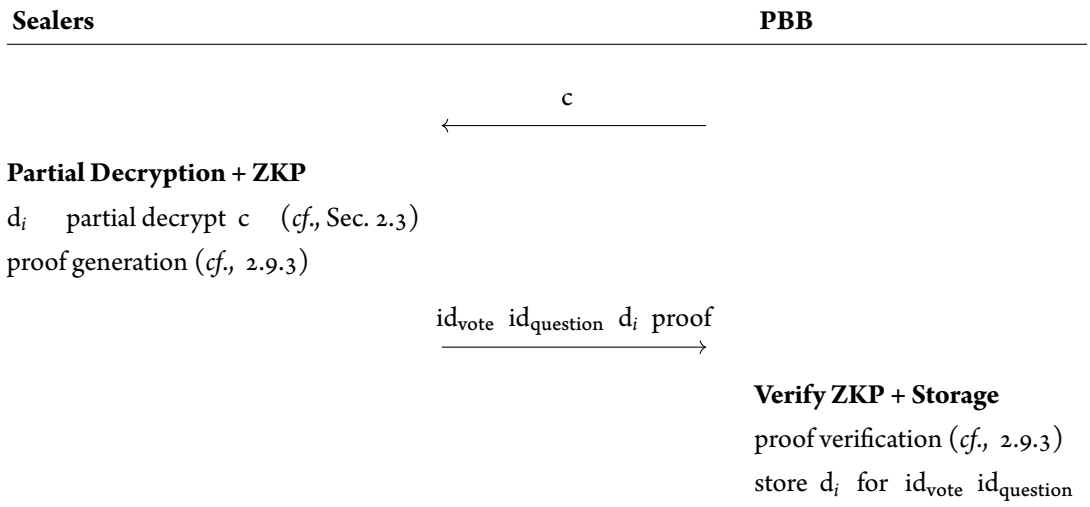


Figure 7.44: Post-Voting Phase - Tallying Step - Ballot Shuffling, Partial Decryption

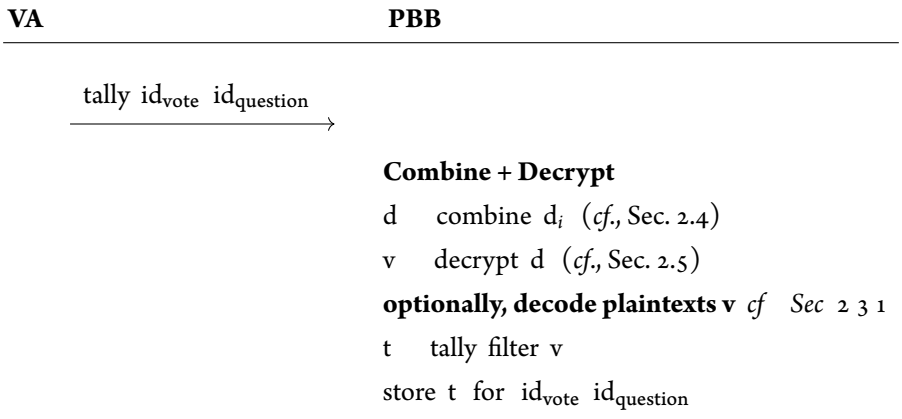


Figure 7.45: Post-Voting Phase - Results - Combine Partial Decryptions, Decrypt and Reveal Plaintexts

8. Results The VA performs the last and final step, combining all the partial decryptions of d_i, the public decryption of the combined partial decryptions d, and optionally, the decoding of the resulting plaintext votes. The step, visualized in Figure 7.45, is initiated by the VA triggering the combination

of the partial decryptions d_i. The DPBB ensures that sealers have submitted their partial decryptions and combines them ballot-wise.

In the last stage, plaintext votes are filtered to ensure any invalid votes **v** or incorrect submissions are not counted. Finally, the filtered set is tallied and for each unique entry, its count is denoted. Once the has been stored and published in the final tally, the voting scheme is finished.

7.4.4 Discussion of Properties

The following sections discuss ProvotuMn from a perspective of its *(i)* privacy and *(ii)* verifiability properties, as well as its *(iii)* practical properties in comparison to its predecessor systems.

Privacy Properties

The following section evaluates and discusses ProvotuMn's privacy properties. Ballot Secrecy (BS), Everlasting BS, Receipt-Freeness (RF), and Coercion-Resistance (CR) are discussed, and Table 7.11 shows the comparison of Provotum 2.0 [179] with the improved protocol of ProvotuMn.

Table 7.11: Comparison of Privacy Properties

	BS	**E-BS**	**RF**	**CR**
Provotum 2.0 [179]				
ProvotuMn 3.0 [172]				

Ballot Secrecy (BS) ProvotuMn achieves BS because ballots are encrypted with the public election key. The DKG used to generate the election key assures that no single trustee can decrypt individual ballots. BS holds, as long as a single sealer remains trustworthy, *i.e.*, does not reveal its vote. Also, the re-encryption randomizer does not access a ballot's content, since the re-encryption homomorphically adds an encryption of zero to the ballot's ciphertext, making it indistinguishable from the input ciphertext. A key assumption of the voting protocol is based on the security of the ElGamal cryptosystem and, therefore, on the DDH assumption [30]. Therefore, E-BS cannot be guaranteed.

Receipt-Freeness (RF) The voter encrypts the ballot using a random value r_1. The randomizer re-encrypts the voter's ballot using another random value r_2. If the voter attempted to prove to a vote buyer that he/she cast the vote in a specific way, it would be impossible, since the randomizer always uses a different random value r_2 in the re-encryption. Thus, only if the randomizer and the voter

would collude, they would be able to prove specific ballot content to a vote buyer. Depending on the threat model and risk assessment, a different kind of randomizer can be employed. In any case, an unauthenticated, non-tappable channel between the voter and randomizer is necessary. Further, in high coercion environments, a randomizer based on a hardware token, as typically used in Internet Banking applications, could be used whereas in other scenarios it might be feasible to deploy a web service.

Coercion-Resistance (CR) Although ProvotuMn achieves BS and RF, it does not achieve CR. A coercer can force a voter *(i)* to give up her voter credentials or *(ii)* to abstain, or *(iii)* force a voter to vote in a specific way. Other countries with established REV systems (*e.g.*, Estonia), circumvent this problem by allowing voters to cast their ballots multiple times, each submission overriding the previous one [289]. Nonetheless, the Swiss law [77] explicitly forbids the repeated ballot casting *i.e.*, each eligible voter is only allowed to cast their ballot once. Thus, making it impossible to guarantee CR in a REV setting in Switzerland without simultaneously providing a physical voting alternative.

Discussion of Verifiability Properties

For a REV system to claim to be End-to-End Verifiable (E2E-V), it must fulfil the individual sub-properties: Cast-as-Intended (CaI), Recorded-as-Cast (RaC) and Counted-as-Recorded (CaR) Verifiability. ProvotuMn achieves these verifiability properties, E2E-V, and also Individual (IV) as well as Universal Verifiability (UV).

Table 7.12: Visualization of Verifiability Properties comparing Provotum 2.0 and ProvotuMn 3.0

	IV	**UV**	**E2E-V**	**CaI**	**RaC**	**CaR**
Provotum 2.0 [179]						
ProvotuMn 3.0 [172]						

Cast-as-Intended Verifiability (CaIV) ProvotuMn implements CaIV as a Challenge-or-Cast, wherein the implementation of the verifier is available in [211]. Challenge-or-cast verification allows voters to either cast their ballot or challenge the encryption mechanisms to test whether the voting client is honest and trustworthy [19]. In this implementation, it is not possible to challenge an encryption and cast the same ballot [195]. If an audited ballot is also submitted, RF and CR will not hold [195]. Thus, to ensure RF and CR, a voter needs to restart from the beginning after challenging

the encryption, and the voting device must not recognize a restart after challenging a ballot [195]. There is also the possibility of re-encrypting the vote with a newly generated randomness instead of starting from scratch again [19, 122].

Individual (IV) & Recorded-as-Cast (RaC) Verifiability After submitting the re-encrypted ballot, the PBB returns a transaction hash digest confirming its inclusion in the DL. As soon as all DL peers reach consensus and the transaction is included in a block, the voter can verify that its submitted ballot has been included.By comparing the ballot from the PBB with the ballot stored on the voter's device, the voter can verify that the ballot has been RaC. This mechanism also fulfills these requirements for IV.

Counted-as-Recorded (CaR) In ProvotuMn, computations and proofs are stored on the PBB and available for public verification. The verification of proofs and computations are included as PBB mechanisms (*e.g.*, combining the public key or the decrypted shares) and can be recomputed manually or by dedicated verification software. A verifier immediately notices manipulation attempts, *e.g.*, if a ballot is replaced during the shuffling process, the verification of the shuffle proof fails. Also, if ballots are invalid, *e.g.*, because the ballot is empty or malformed, this would be recognized after the decryption and before tallying of all plaintexts. All invalid votes are then discarded and excluded from the final tally. Further, the order of ballots, the decrypted shares, as well as the corresponding is inherently given as the PBB verifies the proofs by recomputing all necessary values in the same order as the input values are stored. Therefore, the proof verification always fails if a sealer uses ballots or decrypted shares in a different order during the proof generation than the PBB will use during verification.

Discussion of Practical Properties

The following section evaluates and discusses the properties relevant for the practical deployment of ProvotuMn. An overview of the evolution of practical properties is presented in Table 7.13.

	Fairness	Accountability	Robustness	Transparency	1+	Yes/No	Elections
Provotum 2.0 [179]							
Provotum 3.0 (HE) [148]							
ProvotuMn 3.0 (Mixnet) [91, 172]							

Table 7.13: Visualization of the practical properties across all three Provotum projects.

Fairness In a fair REV system, it is impossible for anyone to compute an intermediary result before the end of the voting period [120]. Thus, fairness is required to avoid anyone can influence the outcome of a vote, by computing a partial tally and distributing it, while the vote is still running. Therefore fairness is provided through applying the DKG process and ensured as long as at least a single sealer remains honest. Thus, it remains impossible to decrypt the ballots prematurely. Only once the voting period has ended are the sealers officially tasked with shuffling and tallying the vote.

Accountability In the ProvotuMn protocol, misbehaving parties can be identified. The protocol realizes *(i)* whether a sealer no longer participates in the DL consensus, *(ii)* if a sealer attempts to alter ballots during shuffling, and *(iii)* whether a sealer did not submit its decrypted share for a particular vote. Due to the fact that n n sealers participate in the DKG need to take part in the distributed tallying, a malicious sealer can only be identified but not excluded from the process without aborting the vote. It is essential for future design iterations that misbehaving parties can be excluded without affecting the outcome of a vote.

The current DKG algorithm requires a threshold of t n, where t n, authorities to be present both during the key generation as well as the Cooperative Decryption (CD). DKG and CD vastly improve the robustness and fairness of a REV system by distributing the trust across multiple nodes in the network and by making early decryption impossible. Nevertheless, the t n threshold bears its risks. It is no longer possible to tally a vote should a sealer loose access to its private key share or turn rogue and not contribute its partial descriptions. To mitigate this risk, alternative options such as Pedersen's DKG [243] not requiring all participants to be present at all times should be explored. Pedersen's DKG is based on every participant running simultaneously a Feldman's verifiable secret sharing (FVSS) [98] scheme as a dealer and, therefore, avoiding a SpoF by only having a single dealer. FVSS is a widely known extension of Shamir's secret sharing [274] that allows to divide a secret value s across a set of participants t by splitting it into multiple parts.

Robustness Since the ProvotuMn DL is operated as a P2P network with each peer storing the complete state of the ledger, the loss of information is avoided per default. Additionally, the network is maintained and state changes are agreed upon using a majority vote among the authorities. Even if some network participants t $\frac{N}{2}$ were to go offline or fail to take part in the block authoring and validation process, the REV system would continue to function normally.

Votes and Elections Even with Provotum 3.0 HE, only simple binary votes were supported. By replacing the HE-based tallying component of the Provotum REV system with a Mixnet-based

approach, it removed the restrictions on votes being either zero or one. This allows for limited votes where voters select k out of N candidates or multi-way elections where voters select a single candidate out of N possibilities.

In some countries, such as Switzerland, write-ins, *i.e.*, votes containing a handwritten name of a candidate, are allowed during the voting process as part of suggesting additional candidates. This is currently not implemented in the prototype but technically possible. The CHVote specification [126] provides a suitable rerference implementing such a feature.

Scalability For a REV system to be applicable in practice, it not only needs to achieve the posed security, privacy and verifiability requirements but also scale with the number of users. To show that the proposed voting protocol and prototype implementation can be used in a real world vote, we first provide a theoretical runtime analysis by counting the of modular exponentiations (`modExp`) and comparing them with the previous projects. For that reason, the following section provides a quantitative performance evaluation with different benchmarks.

7.4.5 Performance Evaluations

The following section covers three major performance evaluations of ProvotuMn. First, the theoretical runtime performance is evaluated, mainly focusing on the modular exponentiations used in the cryptographic protocol. Second, the evaluation covers local (single-machine) benchmark executions of the protocol, leading to the third part, a decentralized evaluation considering up to 10 sealers (*i.e.*, individual servers) which indicates that the distributed deployment is feasible.

Theoretical Runtime Performance

The performance of cryptographic algorithms is usually compared in terms of their use of modular exponentiation (`modExp`). These operations are computationally most expensive when operating with large (*i.e.*, 2048 bits and more) key sizes [127, 128]. In Table 7.14, the computational cost of casting a single ballot in Provotum 2.0 are compared with ProvotuMn. While Provotum 2.0 required 17 `modExp` per ballot, this work only requires an additional 8 `modExp` over Provotum 2.0.

Ballot Validity ZKP Contrary to HE-based REV systems, in a Mixnet-based scheme, all ballots are decrypted *before* the final tallying step and, therefore, ballot validity is checked in plaintext. Thus, in ProvotuMn voters are not required to generate a ballot validity proof, *i.e.*, the zero-knowledge

Table 7.14: Comparison of Cryptographic Operations between Provotum 2.0 and ProvotuMN in Terms of `modExp`.

	Provotum 2.0		**ProvotuMn 3.0**			
Step	Voter	DL	Voter	DL	Randomizer	Sealer
Ballot Encryption	3		2			
Ballot Validity ZKP Generation	6					
Re-Encryption ZKP Generation					2	
Re-Encryption ZKP Verification			2			
Ballot Validity ZKP Randomization						
Ballot Validity ZKP Verification		8				
Ballot Shuffling						2
Shuffle ZKP Generation						8
Shuffle ZKP Verification				9		
Subtotal (per Entity)	9	8	4	9	2	10
Total	17			25		

membership proof showing that the vote contained in the ballot is part of a set of allowed votes (*e.g.*, $v \quad 0\ 1$) reducing the `modExp` required by 14 compared to Provotum 2.0.

Ballot Shuffling. The votes are cryptographically shuffled to ensure that there is no connection between an initially cast ballot and the decrypted ballot. This process requires *(i)* offline shuffling, *(ii)* offline proof generation, and *(iii)* online proof-verification by the PBB. This process shifts the computational overhead to the sealer's sides.

Local Benchmark Results

The practical scalability of ProvotuMn is evaluated by executing benchmarks on a single host of the algorithms with a realistic scenario of a vote containing 1,000,000 ballots. The benchmarks were executed on an Ubuntu 20.04 server using a 12-core AMD Ryzen 3900X CPU, base clock of 3.8 GHz, and 64 GB RAM. The results were averaged over 10 executions of each algorithm. In the following, the benchmarks are grouped according to the voting scheme phase they belong to.

Pre-Voting Phase The pre-voting phase does not contain any operations that require scaling higher than a few participants. Thus, no benchmarks have been performed for this phase.

Voting Phase In the voting phase, the ballot randomization, as well as the vote casting, are operations that need to offer scalability. Further, the ballot encoding is evaluated since it is a critical aspect of the final decryption performance.

Ballot Encoding The ballot encoding used in ProvotuMn differs from Provotum 2.0, and the two strategies are quantitatively compared (*cf.*, Table 7.15). Although the encryption is causing an overhead, the performance degradation is negligible since it is only executed once on the voter's device. Additionally, the encoding causes the performance of the decryption to be exponentially more efficient.

Table 7.15: Performance Difference of the two Message Encoding Strategies for p of Size 2 048 *bits* and q $\frac{p-1}{2}$.

Encoding	**Encryption**	**Decryption**		
		m 0 10	m 0 100	m 0 1000
g^m ([65])	0.66 *ms*	4.21 *ms*	17.09 *ms*	144.93 *ms*
m^q mod p 1 ([94])	4.37 *ms*		2.87 *ms*	

Randomizer The re-encryption of a ballot, as well as generating a designated verifier ZKP, are stateless operations; they are no performance bottleneck even if the operations themselves were not efficient. Additional load can simply be handled by scaling up to more randomizer services.

Ballot Casting The performance of the ballot casting is relevant as it is crucial for the number of incoming ballots the DL network can handle at a time. The time required to process a ballot is only 0.018 *ms* as no processing apart from *(i)* the ballot duplication check, and *(ii)* the randomizer's signature verification is required. Another crucial factor is the number of transactions a DL node can handle per time interval. The ballot casting is benchmarked for the default block time of 6 seconds. The performance of the network, composed of two sealers and a VA, was able to handle 2100 transactions per block. This equates to 350 transactions per second or 1 25 million per hour.

Post-Voting Phase In the post-voting phase, the shuffle ZKP verification and the partial decryption ZKP verification by the PBB need to scale as the shuffling, tallying, and publishing of the results should not take longer than in a regular/postal voting scenario. Therefore, both operations are benchmarked, and the results are discussed in the following. Additionally, the performance of the

ballot shuffling in comparison to the chosen group modulus p is investigated to highlight the impact on the performance of increasing the security level.

Table 7.16: Ballot Shuffling Performance [s] for Different Sizes of p (Group Modulus), *i.e.*, Different Security Levels.

Batch Size	256bit	512bit	1024bit	2048bit	3072bit
10	0.0004	0.002	0.009	0.056	0.19
100	0.005	0.019	0.117	0.757	2.355
1,000	0.047	1.969	1.19	7.524	23.78
10,000	0.474	1.969	11.92	75.3	238.1
100,000	4.736	19.7	119.2	753.1	2382

Shuffle (Crypto Library) The performance of the ballot shuffling operation is compared for the different sizes of the group modulus p (*cf.*, Tab. 7.16). Results indicate that an increase in group modulus p leads to a significant increase in runtime. For example, decreasing the size of p from 2048bit to 256bit would allow for a performance increase of 150 times depending on the batch size. On the other hand, increasing p from 2048bit to 3072bit results in an increased runtime of 3 15 times. An option to allow decreasing the group modulus size while keeping or increasing the level of security is by performing all operations in a group $_q$ defined over an Elliptic Curve (EC). The performance gains are most likely less than the loss shown above, since EC operations are computationally more expensive. Nevertheless, the performance improvements would still be significant.

ZKP Verification. The major bottleneck of ProvotuMn verification, which needs to be performed by the PBB *i.e.*, the DL nodes. Since the sealers alternate with block production within fixed time intervals, conceptually, the DL is a single unit of computing. Thus, the direct performance of the PBB operations directly correlates to the hardware capacity on which the consensus nodes operate. Therefore, the shuffle ZKP verification and partial decryption ZKP verification performance is further analyzed.

Shuffle Proof The runtime of the shuffle proof generation and verification (*cf.*, Tab. 7.17) is comparable with the results of the package benchmark for p 2 048 (see 7.17). Additionally, shuffling as well as shuffle proof generation are inherently parallelizable as these operations are performed offline *i.e.*, the workload can be distributed across all sealers and an increase in ballots can be accounted for by increasing sealers. Therefore, the performance of the shuffle proof verification is the

Table 7.17: Mixnet Performance [s] for p of Size 2 048 *bits*.

Batch Size	Shuffling	Proof Generation	Proof Verification
10	0.07	0.25	0.31
100	0.74	2.27	2.98
1,000	7.47	22.37	29.67
10,000	74.68	223.40	296.50
100,000	746.80	2234	2965

only potential bottleneck, since it is performed on the PBB by the DL network participant authoring the current block. In the current setup, this operation takes 22 3 *ms* per ballot which equates to 6 2 hours of shuffle ZKP verification for 1 000 000 ballots. While not implemented in ProvotuMn as of now, Substrate 3.0 introduced runtime worker threads[238] allowing a Substrate node to process mutually exclusive transactions that do not alter the storage, in parallel.

Table 7.18: Decryption Proof Performance [s] for p of Size 2 048 *bits*, Shows the Duration in Seconds With Varying Batch Sizes, *i.e.*, the Number of Votes in an Operation.

Batch Size	Partial Decryption (PBB)	Partial Decryption (Sealer)	Proof Generation	Proof Verification
10	0.007	0.039	0.042	0.053
100	0.065	0.385	0.388	0.487
1,000	0.696	3.888	3.861	4.858
10,000	6.398	39.093	38.850	48.891
100,000	63.896	391.012	388.628	488.950

Partial Decryption Proof Table 7.18 shows the performance of the partial decryption (Sealer), the ZKP proof generation and verification, and the partial decryption (PBB) for a single substrate node. Similar to the ballot shuffling, the partial decryption (Sealer) and proof generation are performed offline and are, therefore, serializable. Thus, the proof verification and partial decryption (PBB) are the key operations, potentially posing a bottleneck, performed on the PBB by the DL network participant authoring the current block. In the current setup, the proof verification takes 4 8 *ms* per ballot which equates to 1 3 hours for 1 million ballots. Even though this does not directly pose a performance bottleneck, runtime worker threads could again be leverage to parallelize

the proof verification. The performance of the partial decryption, performed on the PBB, does not pose a bottleneck as it only takes 0 6 *ms* per ballot which equates to 0 18 hours for 1 million ballots.

Distributed Deployment Evaluation

ProvotuMn is based on a docker-based architecture provided in [295] and includes Terraform and Ansible scripts to automate its deployment. The distributed evaluation focuses on the major bottlenecks as identified in the theoretical runtime analysis: the mixing. Runtime tests were conducted in a distributed cloud deployment of ProvotuMn. Tests with two, five and ten sealer instances were performed. Each instance was provisioned by DigitalOcean using 2 CPU cores and 2 GB of RAM.

In addition, the block time was fixed to 6 s, which is the time interval in which new blocks are generated in the DL. The mixing batch size is fixed to 75, and the rounds of mixing to 3. The batch size describes how many ballots are shuffled by a sealer in one operation. Each experiment was run 3 times to calculate the average runtime. The runtime is measured by the monitoring system of the infrastructure and contains the time from the start until the end of the mixing step. This means it is an end-to-end experiment that includes all steps necessary to mix the ballots. These steps include the shuffling, the generation, the verification of the shuffle proofs, and the complete interaction with the DL (*e.g.*, storing, reading ballots).

Table 7.19: Mixing Times [s] in the Distributed Deployment

Number of Votes	2 Sealers	5 Sealers	10 Sealers
100	99	91	111
1'000	1'280	1'580	1'750
10'000	10'002	11'535	13'405

Table 7.19 shows those results obtained from the scalability experiment. The data is separated into series of 2, 5, and 10 sealers. The table presents the average runtime in seconds for the mixing step for different numbers of cast votes. The runtime grows linearly in the number of votes for each data series, which indicates that the distributed system scales for large elections. However, the mixing for one vote takes approximately 1 second. Secondly, the increase of the runtime with additional sealers is small. Increasing from 2 sealers to 5 sealers varies between -9% and 23%, and from 5 sealers to 10 sealers varies between 10% and 22%. However, these results should be treated with caution as they are end-to-end tests and high variance can occur, for example, due to network delays. Additional results for the runtime of randomizing votes is depicted in Table 7.20.

Table 7.20: Runtime [s] for randomizing the ballots

Number of Votes	2 Sealers	5 Sealers	10 Sealers
100	20	19	22
1'000	130	212	175
10'000	1'375	2'019	1'976

In conclusion, comparing these results to benchmark results from the single-instance deployment of ProvotuMn (*cf.*, Section 7.4.5), it is inferable that the mixing time in the cloud deployment is around 16 to 26 times slower than the mixing in the original work. The main reason for that lies in the communication overhead and the end-to-end characteristics of this evaluation. As the block time of the DL is 6 seconds, the sealers need to wait until transactions of the previous shuffling were committed. When the shuffling is fast, it will still take 6 seconds until the next sealer can start shuffling. To mitigate this problem, the block time or the batch size could be adjusted. A shorter block time would aggregate transactions faster to the DL's ledger, and a larger batch size would increase the number of votes shuffled by a sealer in one transaction. Another aspect is that the mixing runtime does not increase linearly with the number of sealers. The sealers should sequentially (one after another) shuffle the batch of ballots. This would implicate that a doubling of the sealers should cause a doubling of the runtime. The reason for that behavior stems from implementation issues that need to be addressed in future iterations.

7.5 Æternum

REV system must provide ballot privacy and verifiability of the final tally. Research has proposed REV schemes offering ballot privacy based on computational intractability assumptions (*e.g.*, Provotum, as previously described), but only a few provide Unconditional Privacy (UP). Æternum is a REV system with a voting scheme providing UP. Æternum does not require trust in a central authority, nor does it assume the computational intractability of an underlying mathematical problem to provide UP. To satisfy UP's minimal trust assumptions, Æternum uses a permissioned Distributed Ledger (DL), that forms a decentralized network of permissioned nodes, which serve as a transparent, tamper-proof Decentralized Public Bulletin Board ( ).

7.5.1 System Design

The Æternum System Design contains the description of *(i)* key assumptions of the voting scheme and *(ii)* of the voting scheme for Æternum itself, which lays the basis for achieving Unconditional Privacy (UP).

Assumptions

A REV system is complex and the approach developed enables UP. Thus, the system's architecture impacts design decisions [41] as assumed and formulated within Table 7.10, complemented by Figure 7.46 to illustrate dependencies of these assumptions $A1$ to $A4$. $A1$ assumes the existence of an Identity Management (IdM) system, which allows VAs to authenticate users and generate a list of eligible voters ($U_{eligible}$). Depending on the use case, the IdM is performed by a TTP that is either a governmental authority for votes and elections or, in a corporate setting, an external IdM provider. $A2$ assumes anonymous communication channels, which are required to achieve UP. Quantum-safe communication channels (*e.g.*, Quantum-Safe TLS [108, 234]) do see prototypical implementations available, but are still not ready for practical deployment. $A3$ is given by the current Swiss regulation, which does not allow for multiple votes [81]. $A4$ refers to cryptographic generators that need to be set up by the VA.

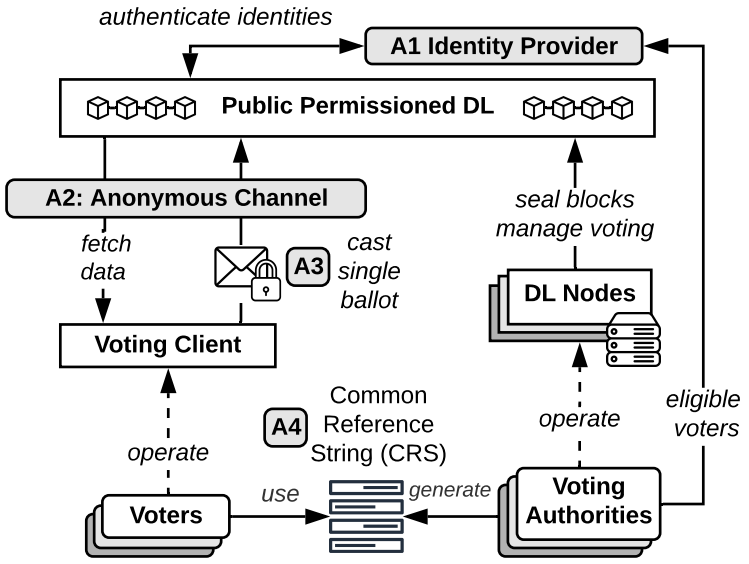


Figure 7.46: Dependencies of Æternum's Assumptions A1 to A4

Table 7.21: Æternum's Assumptions

ID	Description
$A1$	Every eligible voter is uniquely identifiable.
$A2$	The communication between the voter client and the DPBB is over an anonymous channel.
$A3$	Every voter is only allowed to cast a single vote once.
$A4$	Initial trust exists for the VA to set up independent generators.

Voting Scheme

Æternum utilizes the voting scheme from [204] (*cf.*, Fig. 7.47), which does not impose any specific encoding for ballots, since they are not encrypted. It contains four protocols and the main stakeholders are VAs, voters, and the DPBB. The following notations are used to specify the major basic constructs, while the full protocol's details are available in the source code [173] and [204].

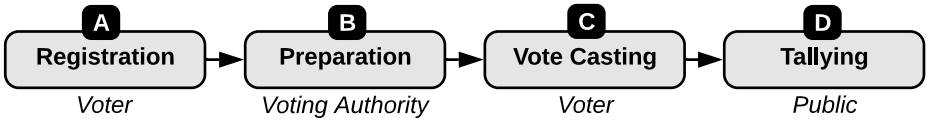


Figure 7.47: The Four Protocols of the UP Scheme [203, 204]

Registration Protocol (Voter) First, (1) a voter V picks the private credentials a β at random from $_q$ and is able to (2) compute a public credential u $h_1^a h_2^\beta$. $_q$ is the set of integers modulo q. In the third step, (3) the public credentials u are posted to the DPBB, registering V for the vote [203].

Registration Protocol (Voter V)

1 Pick Private Credentials a β \$ $_q$

2 Generate public credential u $h_1^a h_2^\beta$

3 Post v u to PBB

Preparation protocol (Voting Authority) First, (1) *VA* verifies the eligibility of the registered voters and compiles a list ($U_{eligible}$), which links eligible voters to their public credential u_i, where N is the electorate size. As mentioned in assumption $A1$, a suitable IdM is required, which allows the VA to verify a voter's eligibility. Second, (2) from the set of public credentials, a polynomial

$P\ X$ $\prod_{i\ 1}^{N} X$ u_i is calculated. $P\ X$ is required for the proofs generated later in the protocol. The polynomial's coefficients A a_0 a_N are posted to the DPBB along with the list of voters U. Anyone can verify the correctness of the polynomial by recalculating it from U. Third, (3) the VA chooses an election generator h $_q$ and (4) finally posts it to the PBB.

Preparation Protocol (Voting Authority *VA*)	
1	Define $U_{eligible}$ $V_1\ u_1$ $V_N\ u_N$
2	Calculate polynomial $P\ X$ $\Pi_{i\ 1}^{N} X$ u_i
3	Select generator h
4	Post $U\ A\ h$ to PBB

Vote Casting protocol Before starting the vote casting stage, the voter checks whether he/she is included in the list of eligible voters. If yes, the voter (1) selects a vote v and (2) generates an election credential u h^{β}. This is crucial since the voter does not authenticate with the public credential u, but instead, the voter sends u with the ballot. This step actively removes the link between the voter's identity u from the vote and requires the voter to generate three Non-Interactive Zero-Knowledge Proofs (NIZKP) [204].

Then, two Pedersen commitments [244] are computed: (4) a commitment c $com_p\ r\ u$ to the public credential u is created, using random r $_R$ $_p$, and also, commitment d $com_q\ s\ a\ \beta$ to the private credentials is computed, with s $_R$ $_q$.

In (5), three NIZKPs are computed. First, a set membership proof π_1 is generated, proving that c is a commitment to one of the credentials in the list of voters. Second, commitments d and c are used to create the second NIZKP π_2, proving knowledge of the private credentials a and β that were used to generate u. Finally, the voter needs to prove that the same β was used to generate the election credential u, which was used in the generation of the public credential u. In this regard, π_3 uses a proof of equality of discrete logarithm, thus preventing the voter from sending multiple ballots with different u [204].

Finally, (6) the ballot tuple B $v\ u\ c\ d\ \pi_1\ \pi_2\ \pi_3$ is posted to the DPBB using an anonymous channel that does not allow linking its ballot to it. The vote itself can be added in plaintext to the ballot in any format. However, remember that voters need to compute NIZKPs dependent on v. When computing the challenge for these NIZKPs, the vote is part of the input into the hash function and impacts the computational effort. Using the vote as input ties v to B because it is used as input again when verifying proofs. If a different v is sent within B, the proof verification will fail.

Vote Casting Protocol (Voter V)

1. Select vote v
2. Compute election credential u h^{β}
3. Pick r $_p$ and s $_q$
4. Compute commitments
   - c $com_p\ r\ u$
   - d $com_q\ s\ \alpha\ \beta$
5. Compute NIZKP
   - π_1 NIZKP $u\ r$ c $com_p\ r\ u$ $P\ u$ 0
   - π_2 NIZKP $u\ r\ s\ \alpha\ \beta$ c $com_p\ r\ u$ d $com_q\ s\ \alpha\ \beta$ u $h_1^{\alpha} h_2^{\beta}$
   - π_3 NIZKP $s\ \alpha\ \beta$ d $com_q\ s\ \alpha\ \beta$ u h^{β}
6. Post ballot B $c\ d\ v\ u\ \pi_1\ \pi_2\ \pi_3$ to PBB

Tallying protocol After the vote casting is concluded, the tallying protocol is executed. Since all data on the PBB are public, anyone can fetch all ballots $\mathcal{B}$ (1), and then verify the proof of each ballot (2). If duplicate votes are allowed, they need to be cleaned at this point (3), *e.g.*, by simply dropping all but the last ballot corresponding to the same u, and (4), the final tally can be accumulated from the plain text votes.

Tallying Protocol

1. Retrieve set $\mathcal{B}$ of all ballots from PBB
2. For each B $\mathcal{B}$ verify $\pi_1\ \pi_2\ \pi_3$
3. Detect duplicates based on u and resolve
4. Compute final election result

Extensions Toward Receipt-Freeness and Fairness

Through dedicated extensions of the basic scheme presented in Section 7.5.1, Æternum can achieve Receipt-Freeness (RF) and Fairness [205]. However, Æternum's assumptions need to be extended

as well: *during* vote casting and tallying, trusted authorities and computational intractability assumptions are required to prevent the creation of invalid ballots. However, these assumptions are not required to guarantee privacy.

Additionally, the voting scheme needs to be adapted in the casting and tallying protocol. In this extended scheme, if all trusted authorities collude and publish their secret values, voters would be able to obtain a receipt, though privacy would still be given. Equally, computational intractability assumptions are only required to prevent the creation of invalid ballots *during* the voting period and to prevent voters from creating a receipt, but not to protect long-term privacy. These extensions of Æternum are based on [205] and apply mix-nets and homomorphic tallying.

During vote casting, the extended version of these Æternum protocol extensions require the voter to create an electronic ballot consisting of the encrypted vote, a commitment to the public credential, an encryption of the election credential, and a composition of zero-knowledge proofs of knowledge of the representation of one of the registered public vote credentials. During tallying, trusted authorities verify proofs included in these ballots, shuffle the list of valid votes in a mix-net, decrypt election credentials, and accumulate votes per voter homomorphically. These accumulated votes are shuffled in another mix-net and decrypted to plaintext.

RF is achieved by not revealing the number of ballots cast by a single voter. Hence, the voter can obtain a receipt for every ballot and potentially hands these over to the vote buyer. However, the vote buyer is not able to determine if the number of receipts obtained is complete. Therefore, the voter can submit an additional ballot that cancels out or overrides all previous ballots [203]. This changes assumption $A3$, since it allows for multiple votes. While the extended Æternum protocol still provides UP, these voting scheme extensions impose additional complexity and stronger assumptions would render Æternum less efficient, since they would require trusted authorities during the vote casting and tallying phases.

Extensions Toward Coercion-Resistance

In order to support CR, Æternum's voting protocol also requires changes of the basic scheme [206] by adapting assumptions since achieving RF is required. Thus, the registration and election preparations are identical to the basic scheme described in Section 7.5.1. After the registration protocol's completion, the voter holds a public credential and the election administration defines the list U_{eligible} and the election generator h during the election preparation.

Thus, the basic scheme changes during the vote casting phase, where the third NIZKP is used [206]. If it is required that the voter is allowed to cast multiple votes, assumption $A3$ changes. In addition, the tallying protocol must guarantee that only the last ballot per voter is considered in the final tally.

Therefore, the protocol detects and excludes updated votes in a verifiable manner (based on mixnets) without leaking anything to a coercer [203].

7.5.2 Implementation

Æternum is implemented as the `up-voting-system` Go module available on GitHub [226]. Æternum's implementation is focused on the basic scheme as described in Section 7.5.1, without the extensions for RF and CR (*cf.*, Sections 7.5.1 and 7.5.1), since otherwise *(i)* these extensions would increase the complexity of the scheme and, thus, worsen computational scalability and *(ii)* Swiss legislation only allows for a single vote to be cast [84].

Each of the three `cli` packages contain a Command Line Interface (CLI) application. The entry to the PBB application is contained in the `cli/pbbd` package, *i.e.*, compiling it generates an executable that can be deployed on a PBB node. The other two packages `cli/vcli` and `cli/acli` compile to a voter and a voting administration CLI application respectively. All three `cli` packages rely on the `pbb` package that implements the PBB and functionality for CLI interaction. Finally, the `crypto` package implements the cryptographic primitives required by the Æternum voting scheme, as described in Section 7.5.1. The `pbb` package implements a Cosmos-SDK module (following a similar approach as *e.g.*, the `staking` module [63]).

System Architecture

Details of the Æternum architecture are depicted in Figure 7.48. The decentralized PBB comprises a DL network of *(i)* consensus nodes and *(ii)* full nodes. Full nodes store a copy of the complete DL and expose web endpoints for clients to post and query. Consensus nodes additionally participate in the consensus mechanism by creating new blocks and validating transactions. In the case of an election or voting, consensus nodes should be operated by trusted authorities, which may not necessarily trust each other, while anyone can operate a full node and directly observe the voting process. The appropriate selection of stakeholders that serve as consensus nodes is based on the use case (*i.e.*, the vote or election at hand). For instance, in the Swiss case, consensus nodes would only be operated by trusted authorities, such as cantons or the federal government, whereas anyone interested could operate a full node.

Although the prototype implementation uses Tendermint's consensus mechanism (requiring at least $\frac{2}{3}$ of all consensus nodes to behave honestly), a suitable alternative is Proof-of-Authority [175, 179]. For instance, in Switzerland, cantons, down to municipalities, can participate in the consensus, only requiring at least $\frac{1}{2}$ nodes to behave honestly. Æternum's voting protocol is agnostic to its net-

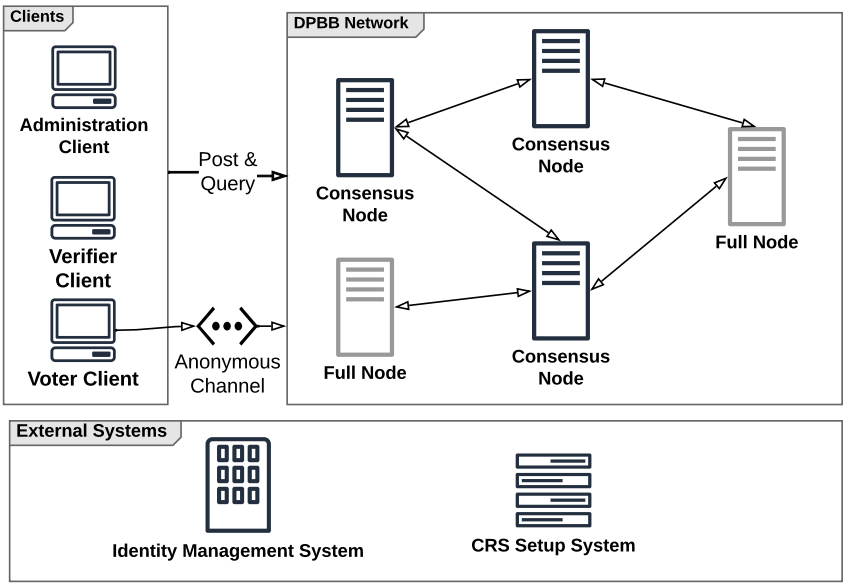


Figure 7.48: Æternum System Architecture.

work persistence layer and thus can be implemented on other BC or DL platforms. However, due to the high complexity of the NIZKPs, access to low-level cryptographic primitives was crucial for implementation. Hence, the highly customizable Cosmos-SDK was more suitable than, for instance, an Ethereum Virtual Machine (EVM) based platform, which is optimized for permissionless usage (*e.g.*, limitation to 256 bit unsigned Integers) [53].

Administration Client The administration client is used by the VA *(i)* to generate the election configuration and post it on the DPBB, *(ii)* authenticate voters' identities that were posted during the registration phase, *(iii)* and then add authenticated, eligible voters to the list on the PBB, *(iv)* compute polynomial coefficients and post themn to the DPBB, and *(v)* define the election generator and post it to the DPBB.

Voter Client The voter client allows voters to *(i)* generate private and public credentials and post the public credential to the DPBB, *(ii)* check the list of eligible voters for the inclusion of their own public credential, and *(iii)* generate and send all necessary parts of a ballot (election credential, commitments, ZKPs, and vote) to the DPBB.

Verifier Client The verifier client allows anyone to verify the correctness of the credential polynomial and the final voting result. Verification is done by re-calculating the polynomial and the elec-

tion result and comparing it to the ones published on the DPBB. The verifier client can *(i)* calculate the credential polynomial from the list of eligible voters and compare it to the published polynomial, and *(ii)* verify all ballots, calculate the election result and compare it to the published result. While the prototype implementation does not provide a separate verifier client, the functionality is implemented in the voter client.

External Systems A suitable Identity Management system (IdM) and a CRS setup system are external systems. The IdM provides Æternum with information about the eligibility of voters (*cf.*, Section 7.5.1, *A*1), while the CRS setup system is an abstraction of the mechanism needed to generate a CRS as a basis for the group generators (*cf.*, Section 7.5.1, *A*6).

Implementation Limitations

While Æternum's voting scheme only requires a valid signature from the voter during the registration protocol (*cf.*, 7.5.1), Tendermint does not allow for unsigned transactions to be included in the BC. However, adding the voter's signature on the ballot transaction would break BP. Therefore, the current implementation of Æternum uses a public and shared keypair to sign these transactions. This fulfills the technical requirement for a valid signature, but, at the same time, keeps the voter's identity hidden. This limitation can be changed by using a different DL that allows for unsigned transactions.

7.5.3 Performance Evaluation

The evaluation of Æternum addresses measurable dimensions in the context of performance aspects focusing on practical facets, especially processing and handling times of the set of solid mechanisms embedded. The ballot transaction is the central transaction for Æternum because it contains the vote. The computations related to the ZKPs affect both the voter client and the DPBB. The voting client generates the NIZKPs, and the DPBB verifies them.

Although the voting scheme [204] does not specify that the proof transcripts have to be verified by the PBB, validating transactions avoids the PBB getting stuffed with invalid ballots. The performance of the NIZKP generation (*cf.*, Table 7.22) and verification (*cf.*, Table 7.23) was evaluated with three different scenarios (averaged over 100 executions) with varying electorate sizes N of 100, 1,000, and 10,000. Those numbers were selected considering the typical distribution of citizens within cantons in Switzerland, with the average number of residents per municipality being 3,880 [40].

In each scenario, the required number of voter credentials were created and registered on the DPBB. Then, the credential polynomial was fetched from the DPBB, and ballots were created for

each voter. The protocol parameters were chosen to provide a security level of 80 (see Section 7.5.4). For performance measurement, two consensus nodes, one for the voter and one for the VA, were deployed as Docker containers. Voter and VA clients were launched on the same machine for all transactions. The performance tests were run on a virtual machine with eight 2.4 GHz Intel Xeon E3 12xx CPUs, 16 GB RAM, running on Ubuntu 18.04.4. Tendermint recommends at least 2GB RAM and a 2 GHz CPU with two cores.

Table 7.22: Average Proof Generation Time [ms]

Electorate Size N	π_1	π_2	π_3	Total
100	105	513	1.66	619.66
1,000	292	581	1.65	874.65
10,000	1,574	579	1.63	2,154.63

Table 7.23: Average Proof Verification Time [ms]

Electorate Size N	π_1	π_2	π_3	Total
100	64	431	2.24	497.24
1,000	115	479	2.36	596.36
10,000	243	482	2.57	727.57

Table 7.22 depicts the average time for the NIZKP generation per voter. Proof π_1 is the set membership proof, π_2 is the proof of known representations (of committed value) and π_3 is the proof of known equality (of discrete logarithms). Effort spent on π_3 is negligible, whereas proof π_2 takes a constant amount of time, largely unaffected by the increased electorate size. As mentioned, π_2 is only dependent on the security parameter κ. Increasing the security level of π_2 by increasing κ will make the time needed for proof generation grow linearly with κ.

In Table's 7.22 column *Total*, results for the proof generation of all three proofs are compared to the results achieved in [204]. There, the poof generation took 1.4, 1.6, and 3 seconds for an electorate of 100, 1,000, and 10,000, respectively. Both UniCrypt and Æternum do not apply any specific optimizations to the proof generation. In [16], an optimized version of the proof π_1 is evaluated, which achieves significantly lower execution times. However, the evaluation is based on a smaller order p for $_p$ of 256 bit and a larger modulus o of 1,536 bit, compared to 1,024 and 1,034 bit with Æternum's evaluation.

only a detailed security analysis can refine requirements posed initially in the design. Thus, this includes Æternum's threat model in brief, an analysis of protocol parameters, and security levels. The evaluation of proof verifications, insecure voter platforms, and malicious participants does reveal risks and threat events, which are all mitigated.

Adversary Model

The adversaries are differentiated into a *present* and a *future* adversary. A global and unbounded adversary is not considered here but considered in [108]. The threat model is congruent with one of the utilized voting scheme [203]. The *present* adversary is active during the voting period *i.e.*, at the time an election occurs. A *present* adversary is bounded by today's available computational resources. Thus, it is assumed that the present adversary is computationally bound. For instance, the adversary cannot efficiently factorize primes, and thus cryptographic primitives based on prime factorization are considered secure. In contrast, the *future* adversary has access to future, much more powerful computational resources *i.e.*, may have access to a Quantum Computer (QC) [108], which will make today's infeasible problems possible to solve.

Æternum provides UP against future adversaries. However, the verifiability properties *during a voting period* rely on the hardness of finding α and β for some $h_1^{\alpha}h_2^{\beta} \quad u \quad U$ which is impossible for a present adversary because of the discrete logarithm assumption. The other option is to construct a proof without knowing α and β. Still, that is impossible for a present adversary because the proofs are computationally sound.

Protocol Parameters

Æternum's voting scheme relies on the definition of the two groups $_p$, $_q$, and their generators, which have to be defined and published on the DPBB before execution of the registration protocol. The VA can publish the generators as long as it is guaranteed that the relative discrete logarithms of the generators are not known to anyone *i.e.*, the generators are independent, otherwise the Pedersen commitments are not binding. For example, by knowing the relative discrete logarithm $x \quad log_{g_1} g_2$, one can come up with an alternative opening $\alpha \quad \beta$ for commitment $com_p \ \alpha \ \beta \quad g_1^{\alpha}g_2^{\beta}$ by replacing g_2 with g_1^x, giving $com_p \ \alpha \ \beta \quad g_1^{\alpha} \ g_1^{x \ \beta} \quad g_1^{\alpha}g_1^{x\beta} \quad g_1^{\alpha \ x\beta}$ and then finding $\alpha \quad x\beta \quad \alpha \quad x\beta$. For instance, an adversary breaking this assumption is able to cast multiple valid votes with only one registered public credential because it would be possible to generate multiple valid private credentials corresponding to that public credential.

Therefore, Æternum does not require trust in a single party, but generators have to be instantiated in a distributed manner. A possible solution is based on a Common Reference String (CRS) from which generators can be derived. However, producing a CRS without trust in one entity introduces more complexity, as can be seen at the example of a multi-party protocol being designed for the setup of a CRS for ZCash [32]. Nonetheless, such a multi-party ceremony might be necessary for a voting system to fully achieve its privacy properties. However, in its current implementation, Æternum does formulate with A4 (*cf.*, Table 7.10) that initial trust in the VA exists to perform the CRS generation.

Security Level

Æternum's verifiability depends on computational intractability *only* during an ongoing election. Thus, protocol parameters have to be configured such that a current adversary is unable to break the scheme during the vote-casting period. In the case of weak parameters, an adversary can find α and β such that for some public credential of an eligible voter u $g_1^{\alpha}\, g_2^{\beta}$ is valid [203]. With these credentials, the attacker can post a ballot, *i.e.*, produce valid NIZKPs, like any other eligible voter. Therefore, the modulus and order of the groups $_p$ and $_q$ have to be large enough to thwart such attacks. Also, the security parameter κ used in proof π_2 needs to be large enough to guarantee the soundness of the proof, *i.e.*, the prover must not be able to make a verifier accept a wrong statement.

Table 7.24: Cryptoperiods' Security Level Recommendations

Cryptoperiod	Security Level	Group Order	Group Modulus	κ
Legacy use	80	160	1,024	80
2019 - 2030	112	224	2,048	112
beyond 2030	128	256	3,092	128

Table 7.24 shows the recommendations by the National Institute of Standards and Technology (NIST) [111], comparing different choices for the security parameter κ and the groups' order and modulus. Numbers in the order and modulus columns are denoted in bit lengths. E.g., for a security level of 80, the group modulus should be a prime of 1024-bit length. The order of the group $_q$ determines the security of the private credentials α β. *I.e.,* the size of the order is the size of the keyspace for the credentials. Therefore the order's bit size is derived from recommendations on key size for DL-based cryptosystems [111]. A security level of 80 is no longer considered secure by NIST [14]. However, because Æternum only assumes computational intractability for the duration of an ongo-

ing election, a security level of 80 is probably still enough at the time of writing. Otherwise, NIST recommends using a security level of at least 112 as of 2019.

Timing of Proof Verification

The ballots' NIZKPs are either *(i)* verified when the ballot reaches the PBB or *(ii)* done after the vote casting phase is concluded, and the DPBB stores all incoming ballots without checking the proofs. In *(i)*, the proof verification's computational intensity puts a heavy load on the DL nodes, and the consensus mechanism Denial of service (DoS) attacks are easily performed and hard to mitigate. In *(ii)*, anyone is allowed to post any number of invalid ballots, which allows for flooding the DPBB with invalid ballots. Partial proof verification is not a viable approach. Proofs π_1 and π_2 are both essential in checking if a ballot comes from an eligible voter and if the sender of the ballot possesses the private credentials corresponding to that voter's public credential.

In order to mitigate a flood of invalid ballots and keep them from reaching consensus nodes, full nodes always run the proof verification on ballots before further broadcasting them through the network. If a ballot is found to be invalid, it can be dropped immediately at any node, thereby not amplifying invalid ballot flooding. Still, verification on a valid ballot has to happen at every node receiving it. Relying on the approval of only one node is not acceptable in a decentralized DL setting.

Casting Multiple Ballots

The voting scheme [203] allows voters to cast multiple ballots that are resolved either immediately or later in the tallying phase. The current implementation of Æternum does not allow for this to prevent replay attacks by checking whether the ballot's election credentials were already used in another ballot. Thus, the PBB does not have to run through the full proof verification. It is guaranteed that the eligible voter will be the first to use the election credential $u \quad h^{\beta}$ because only she knows private credential β. Only after a ballot with that u has been posted will an attacker be able to replay ballots with the same u.

Insecure Voter Platform

A crucial aspect of REV systems is considering the voter devices and the Cast-as-Intended Verifiability property [175]. A compromised voter client presents various threat events caused by malware *(i)* blocking vote broadcasting (and not posting to the DPBB), *(ii)* changing the vote selection *(iii)* breaking ballot secrecy by leaking the voter's choice and identity. While threat events *(i)* and *(iii)* are possible with malware outside of the voter client application, threat event *(ii)* requires that the

attacker successfully modified the voter client executable itself. Threat events *(i)* and *(ii)* can both be detected by verifying the DPBB, which requires the voter to use a second device to connect to the DPBB. For threat event *(iii)*, no countermeasures exist in the Æternum voting scheme. Once a voting client device is compromised, the attacker can link any ballot produced with that device to the owner. Protocol extensions presented in [203] can mitigate the problem, but are not yet part of Æternum.

The trust assumptions of Æternum complicate the practical deployment of voter client applications. If the voters accept an application distributed by the VA, blindly trusting that software client's integrity is not optimal, even if it was built from an open-source code base. Verification mechanisms need to be set up that enable anyone to build the voter client and compare the binary to the application distributed by the VA.

Malicious Network Participants

Even though the current consensus of the DPBB requires at least two-thirds of all consensus nodes to be honest (*i.e.*, is BFT-based), individual malicious nodes can cause disruptions. For instance, a malicious node can censor incoming transactions. However, due to the anonymous channel, voters are not identifiable and cannot be targeted individually. Still, in the registration phase, a node can effectively block a specific voter's attempt to register by dropping transactions received from that voter. As a simple mitigation, the voter's client should verify the correct inclusion of the registration transaction. Another possible censorship is to provide false information. A DPBB node can, *e.g.*, respond with false protocol parameters to a voter's requests. In consequence, the voter will create her credentials based on false groups and generators. In the best case, these credentials will be rejected when posted to the DPBB and, in the worst case, might go unnoticed if the public credential is also an element of the correct group. In the latter case, the voter will think she registered successfully, but will not be able to generate a valid ballot in the casting phase, excluding her from the election. While these threats are feasible, a realistic cost or performance evaluation requires the simulation of a real-world deployment with appropriate mitigation techniques, thus simulating the compromise of a consensus node.

However, these issues can be solved by running a local full node of the DPBB, which implements censorship prevention mechanisms as well as query result verification mechanisms. Censorship resistance is achieved by choosing multiple peers over a so-called peer selection algorithm and sending the transaction to each selected peer. Queries, on the other hand, are executed against the locally kept DL state, and thus no trust in remote nodes is needed. The drawback of running a full node instead of connecting to a remote node is increasing the resource (disk storage, bandwidth, CPU) requirements.

Never underestimate the determination of a kid
who is time-rich and cash-poor.

Cory Doctorow, Little Brother, 2008

8

Summary, Conclusions, and Future Research

In an ideal election scenario, voters express their voting choices remotely over the internet and verify each stage transparently and independently. However, there are many challenges, ranging from societal to technological, that hinder the possibility of a remote election. Within the technical scope, the biggest challenge is to provide a solution that strikes a balance between privacy and verifiability embedded within the governmental and legal structure of a given democratic country.

Electronic Voting (EV) is straightforward in an ideal and secure world: voters send their votes to a trusted counter, which tallies the votes and publishes the final result. However, that is far from being realistic when the opposite is the case: entities involved in EV are typically not fully transparent and trusted, and the context is adversarial. In a traditional EV setting, every participant in an election system is potentially malicious, for example, voters selling votes, dishonest election officials, and corrupt equipment vendors [137].

This PhD thesis improves existing voting schemes by proposing approaches that bolster privacy, verifiability, and auditability in REV and RPV systems alike. First, *(i)* a risk assessment and process analysis of the existing Swiss RPV system was performed, followed by *(ii)* the proposal of fully decentralized voting systems based on public permissioned DLs, and *(iii)* their respective quantitative and qualitative evaluations.

8.1 Summary

Initially, this PhD thesis provided an overview of the practical challenges of Electronic Voting (EV) systems, the global heterogeneity of real-world deployments, and the challenges involved in deploying governmental processes at scale. The internet's emergence and cryptographic advances have enabled novel voting scheme approaches. Further, this thesis highlighted the interdisciplinary nature of Remote EV (REV), involving applied cryptography, systems security, and legal and sociological aspects. Next, the general systems and protocol requirements and specific BC- and DL-based voting system requirements were defined. A literature review classified related work on RPV and REV schemes, comparing them with a focus on privacy, verifiability, and their PBB deployment, as well as trust assumptions. Also, the key difference in voting processes was emphasized: whether votes are cast in person (*e.g.*, in a booth) or remotely (*e.g.*, via the internet). Finally, the Swiss Remote Postal Voting (RPV) system was introduced, stressing that many threats in RPV have not yet been mitigated.

Then, the Swiss RPV process flow analysis laid the foundation to answer the key questions *(i)* whether RPV systems are by default more secure than REV, and *(ii)* what exactly a minimal level of security involves, and *(iii)* which security metrics and mechanisms are mandatory? The Swiss postal voting system is highly successful because substantial trust is placed in third parties (including a wide range of governmental authorities, state-owned companies, and various private companies and suppliers). For a regular citizen, the current process is hard to decipher. Thus, this thesis provided a coherent insight into the Postal Voting Process Flow (PVPF) and identified its weaknesses and strengths with practical examples, subsequently creating public awareness nationwide [101, 102, 157, 174].

Following the Swiss RPV risk assessment, Helverify addresses the privacy and verifiability concerns. Helverify is based on STROBE-Voting [22], employing HE and various to achieve Ballot Privacy (BP) and End-to-End Verifiability (E2E-V). In evaluations, Helverifyhas been shown to scale horizontally for the Swiss use-case (*i.e.*, run Helverifypẽr municipality or canton), which is supported by projecting the evaluation results to real-world elections in Switzerland. In detail, for the largest canton of Switzerland (*i.e.*, Zürich), the projected duration to create and tally ballots for a real-world scenario amounts to eight hours to create 954'327 ballots and 38.9 minutes to tally 566'730 ballots [37]. In terms of security, Helverifydĩstributes the trust amongst voters, the VA, Consensus Nodes, and the PBB, effectively lowering the trust assumptions of each entity.

A crucial aspect of the Swiss RPV system is Identity Management (IdM) for citizen data because the final tallies cannot be guaranteed without the assurance that every final vote was cast by an eligible voter. In Proverum, hybrid public verifiability was introduced, combining the notion of administrative verifiability and eligibility verifiability. Further, Proverum *(i)* introduced an approach combining

a private environment based on private permissioned Distributed Ledgers with a public environment based on public Blockchains, thus achieving public verifiability and an immutable audit trail for the public and for the governmental authorities. Then *(ii)* the Proverum architecture was applied to the Swiss RPV system which mitigates threats present in the current system.

After outlining decentralized RPV systems, the focus shifted to decentralized REV systems. Similar to the methodology applied to the RPV systems (threat modeling and risk assessment as a basis for new system designs), a security analysis of Provotum 2.0 served as a basis for the Provotum 3.0 Homomorphic Encryption (HE) system design. Besides issues uncovered in IdM, a major issue identified was the lack of Receipt-Freeness (RF), which was addressed directly in the Provotum 3.0 HE. Further extensions of the Provotum 3.0 prototype led to a Self-Sovereign Identity (SSI) based IdM system, which allows the selective disclosure of cryptographically signed and verifiable credentials. Further, E2E-V can only be claimed when an independent Cast-as-Intended method is provided. To achieve CaI in Provotum 3.0 HE, an additional vote verifier application is provided, thus achieving complete E2E-V.

A key limitation of Provotum 3.0 HE stems from the cryptographic scheme based on HE. For that reason, ProvotuMn replaces the HE-based cryptographic scheme with a Re-Encryption Mixnet (RMN) based approach, allowing for multi-way elections and write-in candidates. Also, the computationally expensive Zero-Knowledge Proof (ZKP) between the voter and the PBB ensures ballot integrity is no longer necessary. On top of that, the remaining computationally heavy operations, such as ballot shuffling, are shifted from the voter to the voting infrastructure. In summary, ProvotuMn achieves E2E-V *(i)* because voters can Challenge-or-Cast their ballots with a separate air-gapped verifier app, *(ii)* because Recorded-as-Cast verifiability is achieved by allowing voters to verify the correct inclusion on the Decentralized Public Bulletin Board (DPBB), and *(iii)* because Non-Interactive Zero-Knowledge Proofs ( ) of correct decryption and shuffling prove mathematically that the voting protocol was executed correctly. While other voting systems rely on a centralized mixing server, ProvotuMn fully decentralizes the RMN based on a public permissioned Distributed Ledger (DL), thus ensuring publicly readable and tamper-proof storage for the election data. Furthermore, ProvotuMn decentralizes trust through the use of a publicly verifiable Distributed Key Generation (DKG), ensuring that election keys are generated in a publicly verifiable manner.

Congruent with Provotum 3.0 HE, ProvotuMn offers a decentralized re-encryption service ensuring RF. Finally, this approach does not contain any single point-of-failure due to the decentralization of both the *(i)* voting protocol and *(ii)* the underlying consensus layer with identical trust assumptions, which assumes at least $\frac{N}{2}$ 1 honest trustees. The DL keeps an immutable record of all ballots submitted, follows the Proof-of-Authority (PoA) consensus, and ensures the integrity and availability

of the ballots stored. Performance evaluations show that the voting scheme is scalable for large-scale voting and that the major performance bottleneck of ProvotuMn is the generation and verification by the DL nodes. However, the results show that the system can scale to nationwide elections and votes, as the computational complexity scales linearly with the number of ballots cast.

REV schemes based on Mixnets or Homomorphic Encryption rely on computational hardness assumptions to achieve BP [158]. Unconditional BP (UP) does not assume computational intractability and eliminates the need for Trusted Third Parties (TTP) [203]. Minimizing trust assumptions and removing the reliance on TTPs for specific tasks within the voting process is a fundamental argument for decentralized voting systems, *e.g.*, for voting in a consortium of industry participants. Instead of relying on TTPs, the UP-based REV system can be fully decentralized. Since most prior work on REV does not achieve UP, they rely on a TTP or base their privacy on a secret shared between multiple trusted entities [203]. To achieve UP, this thesis introduced Æternum. Æternum's voting scheme provides UP and does not rely on a single trusted authority. To achieve UP, the deployment of a decentralized PBB is required. Distributed Ledgers (DL) offer a suitable approach to deploying a Decentralized PBB (DPBB) to achieve such a distribution of trust among authorities [175]. Concluding, Æternum achieves Unconditional Privacy (UP), secured against future adversaries. While other voting schemes' privacy is vulnerable to future adversaries, Æternum provides UP by utilizing the voting scheme from [203]. Æternum only requires strong parameters for verifiability during the voting period. Additionally, Æternum utilizes a Decentralized Public Bulletin Board (DPBB), implemented as a public permissioned Distributed Ledger (DL). A publicly readable DL enables anyone to verify the DL's election data (*i.e.*, the NIZKPs) and compute the final tally. Further, the DL-based architecture allows for a transparent selection of authorized Voting Authorities (VA) (*e.g.*, authorities or consortia members) that participate in the DL network consensus. Furthermore, measurements indicate that the main performance bottleneck of Æternum is the computational effort required to generate and verify the ballots' NIZKPs (π_1, π_2, π_3). Thus, for nation-wide voting and elections, VAs serve as a TTP, and Æternum's trust assumptions may not be needed. Nevertheless, for use-cases without such structures established and without a central, trusted VA, Æternum provides a suitable alternative. A set of stakeholders can set up and run an election without requiring trust in each other for ballot secrecy.

8.2 Review of Research Questions

The solutions described in this PhD thesis were developed to answer the Research Questions (RQ) presented in Section 1.3. Therefore, the thesis contributions lead to the answer to each RQ. Besides

the key contributions of this PhD thesis, additional contributions can be observed in this thesis and its related publications (*cf.*, Section A).

RQ1 How can existing electoral processes (*e.g.*, the Swiss RPV) be quantified and compared to novel voting channels (*e.g.*, a REV system), considering trust assumptions, privacy, and verifiability properties, and the risks involved in both systems?

RQ1 relates to the current electoral processes deployed in Switzerland, considering the procedural heterogeneity due to the Swiss federal structure. In answering RQ1, the risk assessment of the Swiss RPV (*cf.*, Section 5.3) provided an overview of the relevant TEs, with a focus on confidentiality (C), integrity (I), and availability (A). Research contributions of RQ1 analyzed the current Swiss electoral processes, enabling a comparison between voting channels (*e.g.*, RPV and REV). Investigating RQ1 built the foundation for all subsequent RQs because the findings confirmed that there is room for improvement in the current state-of-the-art.

The research underpinned that RPV is not by default *more secure* than a REV system and that the comparison of voting channels (*e.g.*, RPV with REV) requires a detailed formulation of a trust model (including trust assumptions and stakeholders), as well as the process analysis, followed by a risk assessment. The research performed in the scope of this thesis was also recognized by the media [101], with whom the author conducted further investigations into practically deployed systems [102], creating awareness of election security nationwide [157], which was presented at *Swiss Cyber Storm*, a Swiss cybersecurity conference [174]. Besides these practical answers, the formal declaration of trust assumptions for each of the systems (Helverify, Proverum, ProvotuMn 3.0, Provotum 3.0 HE (and extensions), and Æternum), as well as trust boundaries (*e.g.*, *cf.*, Section 6.2.6 in Helverify) allow for an explicit comparison of solutions and properties achieved.

RQ2 How can electoral processes (especially the Swiss) be publicly verifiable while retaining the privacy of the ballot?

RQ2 relates to the core dichotomy of privacy and verifiability in voting schemes. To answer RQ2, all proposed systems (Helverify, Proverum, ProvotuMn 3.0, Provotum 3.0 HE (and extensions), and Æternum) were designed and implemented with the assumptions provided by the Swiss electoral system. All of these solutions are deployable in the Swiss context and improve privacy and verifiability properties measurably compared to the current status-quo of the Swiss RPV, *(i)* given that *at least one* of the trusted authorities remains honest with regards to the collaborative decryption, and *at least half* of the trusted authorities participating in the permissioned DL consensus remain honest.

Further answers to RQ2 are delivered with Proverum (*cf.*, Section 6.1.1), and the SSI extensions of Provotum 3.0 (*cf.*, Section 7.3, highlighting the potential benefits of applying SSI and public permissioned (and permissionless) DLs to store, exchange, and communicate verifiable credentials and manage digital identities. Further, Helverify highlighted that introducing verifiability to the Swiss RPV could be achieved by introducing the Helverify voting scheme. All approaches were evaluated qualitatively and quantitatively, indicating practical feasibility under specific trust assumptions, which do not differ from the existing Swiss RPV.

RQ3 How can trust be decentralized in systems used for voting and electoral processes while remaining legally compliant and economically viable?

RQ3 relates to the dependency on TTPs in currently deployed real-world voting systems. Thus, answering RQ3 involves decentralized systems based on distributed systems (*e.g.*, BCs and DLs) and applied cryptography. Additionally, the legal requirements in each canton, as well as federal legislation, are considered. Furthermore, economic viability is investigated since cost is also an important factor in public services. Overall, public permissioned DLs address a fundamental issue of REV systems' architectures: the strong assumption of a trusted PBB. REV systems require an immutable, publicly accessible, and verifiable audit trail. Hence, a permissioned DL can serve as a PBB for a REV system.

To answer RQ3, the first step was the detailed specification of *(i)* the requirements necessary for REV systems in general (*cf.*, Section 3.1.2), then *(ii)* the necessary protocol requirements (*cf.*, Section 3.1.1), and finally *(iii)* the detailed requirements for BC or DL-based Voting (*cf.*, Section 3.3).

Further, *(ii)* each of the proposed systems (Helverify, Proverum, ProvotuMn 3.0, Provotum 3.0 HE (and extensions), and Æternum) formulate explicit assumptions (*cf.*, Section 7.4.2 in ProvotuMn) and trust boundaries (*cf.*, Section 6.2.6 in Helverify, *cf.*, Section 7.2.5 in Provotum's CaI extension). Generic trust assumptions were also formulated in the scope of the literature review (*cf.*, Section 4.1). All of the proposed systems took the Swiss legislation [81] into consideration, which states that *(i)* only a single vote cast is valid per voter (*i.e.*, CR de-facto not achievable), as well as *(ii)* detailed requirements regarding E2E-V.

To answer RQ3 with regards to economic viability, each of the proposed systems was evaluated in scenarios considering past Swiss elections (*cf.*, Section 6.2.8 in Helverify), or delivering quantitative results in distributed evaluations that prove practical feasibility (*cf.*, Section 7.4.5 in ProvotuMn 3.0).

RQ4 Which architectural components are required to provide privacy and verifiability for voting and electoral processes?

RQ4 seeks to define the major building blocks of a fully decentralized, private, and verifiable voting system. Consequently, RQ4 relates to the software aspects (*e.g.*, software engineering, architecture, deployment). Thus, RQ4 encompasses all previous RQs as its answer depends directly on the specific systems (Helverify, Proverum, ProvotuMn 3.0, Provotum 3.0 HE (and extensions), and Æternum).

To answer RQ4, voting system approaches were first classified and based on that, the atomic components of REV systems were identified (*cf.*, Section 3.2.2). Architectural components also include the clear definition of involved stakeholders, their trust assumptions (*cf.*, RQ3), and their roles in the system and voting scheme.

Thus, the necessary architectural components are *(i)* a public permissioned DL (as PBB) embedding the cryptographic voting scheme, *(ii)* a voting client (for the voter to cast votes), *(iii)* an Identity Provider (formulated as an assumption, or integrated as designed in Section 7.3). Two optional components were also identified: *(a)* an independent, air-gapped vote verifier application to achieve E2E-V (*cf.*, Section 7.2), and *(b* a re-encryption randomizer to achieve RF (*cf.*, Section 7.4.2).

8.3 Future Research Directions

Future work should consider research on the design and implementation of a suitable REV or RPV voting system where the highest possible privacy and verifiability properties are achieved. Although the key RQs on these topics are addressed in this PhD thesis, future work remains to be explored.

The Swiss RPV is a heterogeneous process involving many different stakeholders and TTPs. Apparently, the federal laws to guide the deployment of secure e-Counting tools are not yet complete[241]. Thus, *(i)* the compilation of an open and transparent list of all the electronic tools in use in the current PV flow could help identify further threat events, enabling the design of mitigation measures to handle risks better. Further, *(ii)* the RA could be extended and ultimately applied to real-world processes in cantons. Then, identified TEs can be assessed in more detail and improvements can be provided and compared to deployed EV systems.

Although Helverify's PoC achieves high fidelity, there are some crucial elements to address in order to reach technical maturity for real-world deployment. First, *(i)* refactoring the cryptographic primitives to use ECC instead of finite-field cryptography, which promises performance and storage space advantages. Using ECC might also positively affect scalability regarding the number of options/candidates because of performance improvements when generating NIZKPs. Second *(ii)*, the ballot scanning process could be optimized by implementing an OCR scanning mechanism, replacing the manual inputs by election officials, yielding an increase in ballot processing speed. Addition-

ally, this reduces the manual work an election official must do to register ballot choices. Third, *(iii)* support for write-in elections could be another area of future work, as the current prototype only supports predefined lists of candidates/options.

ProvotuMn, as well as Provotum 3.0 HE, could both be optimized through the *(i)* the adaption of the Elliptic Curve Cryptography (ECC) to increase scalability even further because compared to Finite Field Cryptography, ECC allows smaller key sizes with the same security level. Second *(ii)*, the upgrade to Substrate 3.0 will allow for parallel computations and increase the scalability of all shuffling operations. Lastly, *(iii)* the integration of more flexible thresholds for the DKG (*e.g.*, 3-out-of-5 thresholds) is also possible, which would ensure an even higher level of trust.

Æternum's possible future work includes *(i)* the integration of a suitable anonymous channel, which could be executed within Æternum's DPBB network directly, and *(ii)* the design of a suitable Multi-Party Computation for the CRS setup. Furthermore, *(iii)* performance optimizations of the set membership proof [16] need to be considered for large electorates (*e.g.*, country-wide deployments). Also, *(iv)*, future work includes the extension of Æternum with Receipt-Freeness, as proposed in [207]. Lastly, the current implementation's availability is not bolstered against DoS attacks, and mitigations are possible future work (*e.g.*, Only allowing a single, cryptographically signed vote transaction per u).

Overall, future research directions for EV research are concerned with the increase of cyberattacks and security, which have to be considered from a practical perspective. Even if, theoretically, all requirements and security considerations are deployed, electronic voting systems are not straightforward to deploy in practice due to the high complexity of the security mechanisms (*i.e.*, this requires highly specialized and "trusted personnel"). For that reason, it is fundamental that these systems are auditable to prevent insecure deployment or operational vulnerabilities. In essence, cryptographic voting schemes must always be able to perform error detection and recover the system's state optimally. Therefore, the critical challenges in future research remain unchanged but need to adapt dynamically to new attack vectors to achieve a balance of privacy and verifiability with a minimal level of risk to ensure the integrity of the final tally.

Bibliography

[1] A. Abraham, S. More, C. Rabensteiner, and F. Hörandner, "Revocable and offline-verifiable self-sovereign identities," https://www.republik.ch/2021/01/28/die-probleme-mit-der-schweizer-e-identitaet, 2020.

[2] B. Adida, "Advances in cryptographic voting systems," Ph.D. dissertation, California Institute of Technology and Massachusetts Institute of Technology, 2006.

[3] B. Adida, "Helios: Web-based Open-audit Voting," USENIX 17th Security Symposium (SS08), San Jose, CA, USA, 2008, p. 335–348.

[4] D. R. Alex Preukschat, "Self-Sovereign Identity," https://www.manning.com/books/self-sovereign-identity, December 2020.

[5] S. T. Ali and J. Murray, "An Overview of End-to-End Verifiable Voting Systems," Computing Research Repository (CoRR) in arXiv, 2016. [Online]. Available: https://arxiv.org/abs/1605.08554

[6] K. Andersen, R. Medaglia, R. Vatrapu, H. Zinner Henriksen, and R. Gauld, "*The Forgotten Promise of E-Government Maturity: Assessing Responsiveness in the Digital Public Sector*," Government Information Quarterly, Vol. 28, Issue 4, pp. 439–445, October 2011.

[7] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain," ITASEC, ser. CEUR Workshop Proceedings, Vol. 2058. CEUR-WS.org, 2018.

[8] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption," IACR Cryptology ePrint Archive, Vol. 2015, p. 1192, 2015.

[9] Association for Computing Machinery (ACM), "ACM Digital Library," https://dl.acm.org/.

[10] J.-P. Aumasson, Serious Cryptography: A Practical Introduction to Modern Encryption. No Starch Press, 2017.

[11] J.-P. Aumasson, Crypto Dictionary: 500 Tasty Tidbits for the Curious Cryptographer. No Starch Press, 2021.

[12] S. Bachmann, "Identity Management in a Decentralized Remote Electronic Voting System," 2021, Master Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[13] D. Barben, "Moutier-Abstimmung: Folgt das nächste Desaster?" Mar. 2021, https://pvpf.ch/moutier.

[14] E. Barker, "Recommendation for Key Management – Part 1: General," pp. 1–142, 2016. [Online]. Available: https://pvpf.ch/nist80057

[15] E. Barker and Q. Dang, "Recommendation for Key Management Part 3: Application-Specific Key Management Guidance," National Institute of Standards and Technology, Tech. Rep., January 2015.

[16] S. Bayer and J. Groth, "Zero-knowledge Argument for Polynomial Evaluation with Application to Blacklists," Lecture Notes in Computer Science, Vol. 7881 LNCS, 2013, pp. 646–663.

[17] S. Bell, J. Benaloh, M. D. Byrne, D. Debeauvoir, B. Eakin, P. Kortum, N. McBurnett, O. Pereira, P. B. Stark, D. S. Wallach, G. Fisher, J. Montoya, M. Parker, and M. Winn, "Star-vote: A secure, transparent, auditable, and reliable voting system," 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13). Washington, D.C.: USENIX Association, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/evtwote13/workshop-program/presentation/bell

[18] J. Ben-Nun, N. Fahri, M. Llewellyn, B. Riva, A. Rosen, A. Ta-Shma, and D. Wikström, "A new implementation of a dual (paper and cryptographic) voting system," 5th International Conference on Electronic Voting 2012 (EVOTE2012), M. J. Kripp, M. Volkamer, and R. Grimm, Eds. Bonn: Gesellschaft für Informatik e.V., 2012, pp. 315–329.

[19] J. Benaloh, "Simple verifiable elections," 2006 USENIX/ACCURATE Electronic Voting Technology Workshop, Vol. 6, 2006. [Online]. Available: https://www.usenix.org/conference/evt-06/simple-verifiable-elections

[20] J. Benaloh, "Ballot casting assurance via voter-initiated poll station auditing," Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology, ser. EVT'07. USA: USENIX Association, 2007, p. 14.

[21] J. Benaloh, "Administrative and Public Verifiability: Can We Have Both?" Conference on Electronic Voting Technology (EVT 2008). USA: USENIX Association, 2008.

[22] J. Benaloh, "STROBE-Voting: Send Two, Receive One Ballot Encoding," Electronic Voting, R. Krimmer, M. Volkamer, D. Duenas-Cid, O. Kulyk, P. Rønne, M. Solvak, and M. Germann, Eds. Cham: Springer International Publishing, 2021, pp. 33–46.

[23] J. Benaloh, M. Bernhard, J. A. Halderman, R. L. Rivest, P. Y. A. Ryan, P. B. Stark, V. Teague, P. L. Vora, and D. S. Wallach, "Public Evidence from Secret Ballots," Computing Research Repository (CoRR) in arXiv, 2017. [Online]. Available: https://arxiv.org/abs/1707.08619

[24] J. Benaloh, P. Y. A. Ryan, and V. Teague, "Verifiable Postal Voting," Security Protocols XXI, B. Christianson, J. Malcolm, F. Stajano, J. Anderson, and J. Bonneau, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 54–65.

[25] J. Benaloh and D. Tuinstra, "Receipt-free Secret-ballot Elections," 26th Annual ACM Symposium on Theory of Computing, (STOC 1994), May 1994, pp. pp. 544–553.

[26] Berner Zeitung, "*Stimmzettel fehlen, Nachzählung über Motorfahrzeugsteuern ist gefährdet*," August 2011, http://pvpf.ch/bz-pb.

[27] D. Bernhard, O. Pereira, and B. Warinschi, "How Not To Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios," Advances in Cryptology (ASIACRYPT 2012), X. Wang and K. Sako, Eds. Beijing, China: Springer LNCS, Vol. 7658, 2012, pp. 626–643.

[28] D. Bernhard and B. Warinschi, "Cryptographic Voting - A Gentle Introduction," International School on Foundations of Security Analysis and Design VII (FOSAD 2012/2013 Tutorial Lectures). Springer LNCS, Vol. 8604, 2014, pp. 167–211.

[29] G. R. BLAKLEY, "Safeguarding cryptographic keys," 1979 International Workshop on Managing Requirements Knowledge (MARK), 1979, pp. 313–318.

[30] D. Boneh, "The Decision Diffie-Hellman problem," 3rd International Symposiun on Algorithmic Number Theory (ANTS-III). Portland, Oregon, USA: Springer LNCS, Vol. 1423, 1998, pp. 48–63.

[31] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, 2020, Version 0.5, Jan. 2020 http://toc.cryptobook.us/.

[32] S. Bowe, A. Gabizon, and M. D. Green, "A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK," Lecture Notes in Computer Science, Vol. 10958 LNCS, pp. 64–77, 2019.

[33] M. Bühlmann, "Schweiz am Sonntag, Aargauer Zeitung: *Das Stimmcouvert per Post verschicken - ein Gratisangebot, das viele Aargauer ausschlagen*," http://pvpf.ch/az-ve, February 2016.

[34] B. Bumbacher, "Neue Zürcher Zeitung: *Hauswart fälscht aus Frust Stimmzettel bei Gemeindewahl*," October 2005, http://pvpf.ch/nzzfraud.

[35] Bundesamt für Statistik, "Änderung des Covid-19-Gesetzes," https://www.bfs.admin.ch/bfs/de/home/statistiken/politik/abstimmungen/jahr-2021/2021-11-28/aenderung-covid19-gesetz.html.

[36] Bundesamt für Statistik, "Änderung des Covid-19-Gesetzes, nach Kanton," https://www.bfs.admin.ch/bfs/de/home/statistiken/politik/abstimmungen/jahr-2021/2021-11-28/aenderung-covid19-gesetz.assetdetail.20004979.html.

[37] Bundesamt für Statistik, "Bundesgesetz über die gesetzlichen Grundlagen für Verordnungen des Bundesrates zur Bewältigung der Covid-19-Epidemie (Covid-19-Gesetz), nach Kanton," https://www.bfs.admin.ch/bfs/de/home/statistiken/politik/abstimmungen/jahr-2021/2021-06-13/covid19-gesetz.assetdetail.17484063.html.

[38] Bundesamt für Statistik, "Covid-19-Gesetz," https://www.bfs.admin.ch/bfs/de/home/statistiken/politik/abstimmungen/jahr-2021/2021-06-13/covid19-gesetz.html.

[39] Bundesamt für Statistik, "Stimmbeteiligung," https://www.bfs.admin.ch/bfs/de/home/statistiken/politik/abstimmungen/stimmbeteiligung.html.

[40] Bundesamt für Statistik, "Regionalporträts 2020: Kennzahlen aller Gemeinden," 2020. [Online]. Available: https://bcbev.ch/bfs-gemeinde

[41] J. E. Burge and D. C. Brown, "Software Engineering Using RATionale," Journal of Systems and Software, Vol. 81, No. 3, pp. 395–413, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121207001203

[42] S. Bursuc, G. S. Grewal, and M. D. Ryan, "Trivitas: Voters directly verifying votes," International Conference on E-Voting and Identity. Springer, 2011, pp. 190–207.

[43] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," 2013, https://ethereum.org/en/whitepaper/.

[44] C. Cadwalladr and E. Graham-Harrison, "Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach," https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, 2018.

[45] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, "Beleniosrf: A non-interactive receipt-free electronic voting scheme," Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1614–1625.

[46] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, "BeleniosRF: A Non-Interactive Receipt-Free Electronic Voting Scheme," ACM Conference on Computer and Communications Security (CCS '16), Vienna, Austria, 2016, pp. 1614–1625.

[47] M. Chaieb, S. Yousfi, P. Lafourcade, and R. Robbana, "Verify-Your-Vote: A Verifiable Blockchain-based Online Voting Protocol," 15th European Mediterranean and Middle Eastern Conference on Information Systems (EMCIS 2018). Limassol, Cyprus: Springer LNBIP, Vol. 341, 2018, pp. 1–14.

[48] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," Communications of the ACM, Vol. Vol. 24, No. No. 2, pp. pp. 84–90, February 1981.

[49] D. Chaum, "Blind Signatures for Untraceable Payments," Advances in Cryptology (CRYPTO '82), D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Santa Barbara, California, USA, 1983, pp. 199–203.

[50] D. Chaum and T. P. Pedersen, "Wallet databases with observers," CRYPTO, ser. Lecture Notes in Computer Science. Springer, 1992, Vol. 740, pp. 89–105.

[51] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," Commun. ACM, Vol. 24, No. 2, pp. 84–90, Feb. 1981. [Online]. Available: http://doi.acm.org/10.1145/358549.358563

[52] B. Chevallier-Mames, P.-A. Fouque, D. Pointcheval, J. Stern, and J. Traoré, On Some Incompatible Properties of Voting Schemes. Springer Berlin Heidelberg, 2010, pp. 191–199.

[53] M. Chowdhury, M. Ferdous, K. Biswas, N. Chowdhury, A. Kayes, M. Alazab, and P. Watters, "A Comparative Analysis of Distributed Ledger Technology Platforms," IEEE Access, Vol. 7, pp. 167 930–167 943, 2019.

[54] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward A Secure Voting System," 2008 IEEE Symposium on Security and Privacy (SP '08), Oakland, CA, USA, 2008, pp. 354–368.

[55] J. D. Cohen and M. J. Fischer, "A Robust and Verifiable Cryptographically Secure Election Scheme," 26th Annual Symposium on Foundations of Computer Science (SFCS '85). Portland, OR, USA: IEEE Computer Society, 1985, p. 372–382.

[56] C. A. Cois, J. Yankel, and A. Connell, "Modern DevOps: Optimizing software development through effective system interactions," 2014 IEEE International Professional Communication Conference (IPCC 2014). IEEE, 2014, pp. 1–7.

[57] Computer Security Division, Information Technology Laboratory, "*Minimum Security Requirements for Federal Information and Information Systems*," US Department of Commerce, NIST, (FIPS PUB 200), March 2006.

[58] V. Cortier, J. Dreier, P. Gaudry, and M. Turuani, "A simple alternative to benaloh challenge for the cast-as-intended property in helios/belenios," November 2019, https://hal.inria.fr/hal-02346420.

[59] V. Cortier, F. Eigner, S. Kremer, M. Maffei, and C. Wiedling, "Type-based verification of electronic voting protocols," International Conference on Principles of Security and Trust. Springer, 2015, pp. 303–323, https://doi.org/10.1007/978-3-662-46666-7_16.

[60] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Election verifiability for helios under weaker trust assumptions," European Symposium on Research in Computer Security. Springer, 2014, pp. 327–344.

[61] V. Cortier, P. Gaudry, and S. Glondu, "Belenios: a simple private and verifiable electronic voting system," Foundations of Security, Protocols, and Equational Reasoning. Springer, 2019, pp. 214–238.

[62] V. Cortier and B. Smyth, "Attacking and fixing Helios: An analysis of ballot secrecy," Journal of Computer Security, Vol. 21, No. 1, pp. 89–148, 2013.

[63] Cosmos SDK, "Documentation: Staking Module." [Online]. Available: https://docs.cosmos.network/master/modules/staking/

[64] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," Advances in Cryptology — CRYPTO '94, Y. G. Desmedt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 174–187.

[65] R. Cramer, R. Gennaro, and B. Schoenmakers, "A Secure and Optimally Efficient Multi-Authority Election Scheme," 17th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '97), Konstanz, Germany. Springer LNCS, Vol. 1233, 1997, pp. 103–118.

[66] B. L. Crimmins, M. Rhea, and J. A. Halderman, "RemoteVote and SAFE Vote: Towards Usable End-to-End Verification for Vote-by-Mail," 2021. [Online]. Available: https://arxiv.org/abs/2111.08662

[67] J. Cucurull, A. Rodríguez-Pérez, T. Finogina, and J. Puiggali, "*Blockchain-Based Internet Voting: Systems' Compliance with International Standards*," Business Information Systems Workshops - (BIS 2018) International Workshops, Berlin, Germany, July 2018, pp. pp. 300–312.

[68] A. De Santis, S. Micali, and G. Persiano, "Non-Interactive Zero-Knowledge Proof Systems," Advances in Cryptology: Proceedings of CRYPTO '87, Santa Barbara, CA, USA, C. Pomerance, Ed. Springer LNCS, Vol. 293, 1988, pp. 52–72.

[69] Decentralized Identity Foundation, "DIF - Universal Resolver," https://dev.uniresolver.io.

[70] Decentralized Identity Foundation, "Performant and modular APIs for Verifiable Data and SSI," https://github.com/decentralized-identity/ethr-did-resolver.

[71] Decentralized Identity Foundation, "Universal resolver implementation and drivers," https://github.com/decentralized-identity/universal-resolver.

[72] D. Y. M. del Blanco, D. Duenas-Cid, and H. A. Moretón, "E-voting system evaluation based on the council of europe recommendations: nvotes," International Joint Conference on Electronic Voting. Springer, 2020, pp. 147–166.

[73] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols: A taster," Towards Trustworthy Elections: New Directions in Electronic Voting, D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutylowski, and B. Adida, Eds. Berlin, Heidelberg: Springer LNCS, Vol. 6000, 2010, pp. 289–309.

[74] Der Regierungsrat des Kantons Aargau, "*101 -Bundesverfassung der Schweizerischen Eidgenossenschaft vom 18. April 1999 (Stand am 23. September 2018)*," http://pvpf.ch/bv.

[75] Der Regierungsrat des Kantons Aargau, "*131.111 - Verordnung zum Gesetz über die politischen Rechte (VGPR) in Kraft seit 01.01.2013, Beschlussdatum: 30.05.2012)*," http://pvpf.ch/vgpr.

[76] Der Schweizerische Bundesrat, "*161.1 Bundesgesetz über die politischen Rechte (BPR) (Stand am 1. November 2015),*" http://pvpf.ch/bpr.

[77] Der Schweizerische Bundesrat, "*161.11 Verordnung über die politischen Rechte (VPR) (Stand am 15. Januar 2014),*" http://pvpf.ch/vpr.

[78] Y. Desmedt and P. Chaidos, "Applying divertibility to blind ballot copying in the Helios internet voting system," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 7459 LNCS. Springer, 2012, pp. 433–450.

[79] Die Bundesversammlung der Schweizerischen Eidgenossenschaft, "Bundesgesetz über die gesetzlichen Grundlagen für Verordnungen des Bundesrates zur Bewältigung der Covid-19-Epidemie," https://www.fedlex.admin.ch/eli/cc/2020/711/de.

[80] Die Bundesversammlung der Schweizerischen Eidgenossenschaft, "SR 101 Bundesverfassung der Schweizerischen Eidgenossenschaft vom 18. April 1999 (Stand am 13. Februar 2022)." [Online]. Available: https://www.fedlex.admin.ch/eli/cc/1999/404/de

[81] Die Schweizerische Bundeskanzlei (BK), "*161.116 Verordnung der BK über die elektronische Stimmabgabe (VEleS) vom 13. Dezember 2013 (Stand am 1. Juli 2018),*" https://pvpf.ch/veles.

[82] Die Schweizerische Bundeskanzlei (BK), "*Anforderungskatalog Druckereien fur Vote électronique,*" http://pvpf.ch/bkreq.

[83] Die Schweizerische Bundeskanzlei (BK), "*Vote électronique,*" http://pvpf.ch/ve.

[84] Die schweizerische Bundeskanzlei (BK), "Verordnung der BK über die elektronische Stimmabgabe (VEleS) (vom 13. Dezember 2013 (Stand am 1. Juli 2018))," 2013. [Online]. Available: https://www.admin.ch/opc/de/classified-compilation/20132343/index.html

[85] Die Schweizerische Bundeskanzlei (BK), "Änderung des Bundesgesetzes über die politischen Recht)," Erläuternder Bericht zur Vernehmlassung, December 2018.

[86] Die Schweizerische Post AG, "*Das Briefrestmonopol - Finanzierungspfeiler für die Grundversorgung,*" http://pvpf.ch/spmon.

[87] Die Schweizerische Post AG, "*Factsheet, Briefe Wahl- und Abstimmungssendung,*" http://pvpf.ch/spfs.

[88] Docker Inc., "Docker," https://www.docker.com/.

[89] E-Voting-Moratorium, "*Initiativtext,*" http://pvpf.ch/evmor.

[90] M. Eck. Provotum Mixnet Source Code. [Online]. Available: https://github.com/provotum/provotum-mixnet

[91] M. Eck, "Mixnets in a Distributed Ledger Remote Electronic Voting System," 2021, Master Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[92] M. Eck, N. Zaugg, and A. Scheitlin, "Design and Implementation of Blockchain-based E-Voting," Master Project, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland, 2020.

[93] P. Ekparinya, V. Gramoli, and G. Jourjon, "The Attack of the Clones Against Proof-of-Authority," arXiv, 2 2019. [Online]. Available: http://arxiv.org/abs/1902.10244

[94] M. El Laz, B. Grégoire, and T. Rezk, "Security Analysis of ElGamal Implementations," 17th International Conference on Security and Cryptography. SCITEPRESS-Science and Technology Publications, 2020, pp. 310–321.

[95] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms," IEEE Transactions on Information Theory Vol. 31, No. 4, pp. 469–472, 1985.

[96] S. Emery, C. E. Chow, and R. White, "Penetration Testing a US Election Blockchain Prototype," Electronic Voting. E-Vote-ID 2021., pp. 82–97, 2021.

[97] E. F. Hao, "Schnorr Non-interactive Zero-Knowledge Proof," RFC Editor, Tech. Rep., 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8235

[98] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), 1987, pp. 427–438.

[99] T. M. Fernandez-Carames and P. Fraga-Lamas, "Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks," IEEE Access, Vol. 8,, pp. 21 091–21 116, 2020.

[100] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," Advances in Cryptology (CRYPTO '86), A. M. Odlyzko, Ed. Springer, Berlin Heidelberg, 1987, pp. pp. 186–194.

[101] A. Fichter, "Republik: Passwort: «Wahlen»," 2020, https://www.republik.ch/2020/09/25/passwort-wahlen.

[102] A. Fichter, "Republik: «Passwort: Wahlen» – der technische Hintergrund und das Glossar zur Recherche," 2020, https://www.republik.ch/2020/09/25/passwort-wahlen-der-technische-hinter-grund-und-das-glossar-zur-recherche.

[103] C. Fontaine and F. Galand, "A Survey of Homomorphic Encryption for Nonspecialists," EURASIP Journal on Information Security, No. 013801, pp. pp. 1–10, 2007.

[104] A. Fujioka, T. Okamoto, and K. Ohta, "A Practical Secret Voting Scheme for Large Scale Elections," Advances in Cryptology (AUSCRYPT '92). Gold Goast, QLD, Australia: Springer LNCS, Vol. 718, 1992, pp. 244–251.

[105] A. Fumagalli, "Neue Zürcher Zeitung: *Die GLP getraute sich nicht, den Champagner zu öffnen – weil es der Wahlchef einer 2600-Seelen-Gemeinde vermasselt hatte*," October 2019, http://pvpf.ch/arzier.

[106] J. Furukawa and K. Sako, "An efficient scheme for proving a shuffle," Annual International Cryptology Conference. Springer, 2001, pp. 368–387.

[107] D. Gambetta, "Can We Trust Trust?" Trust: Making and Breaking Cooperative Relations, Vol. 13, pp. 213–237, 2000. [Online]. Available: https://bit.ly/3nAXyRm

[108] H. Ge, S. Y. Chau, V. E. Gonsalves, H. Li, T. Wang, X. Zou, and N. Li, "Koinonia: Verifiable E-Voting with Long-term Privacy," Proceedings of the 35th Annual Computer Security Applications Conference, ser. ACSAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 270–285. [Online]. Available: https://doi.org/10.1145/3359789.3359804

[109] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems," Journal of Cryptology, Vol. 20, No. 1, pp. 51–83, 2007.

[110] R. Gharadaghy and M. Volkamer, "Verifiability in electronic voting-explanations for non security experts," 4th International Conference on Electronic Voting 2010. Gesellschaft für Informatik eV, 2010.

[111] D. Giry and BlueKrypt, "Cryptographic Key Length Recommendation," 2020. [Online]. Available: https://www.keylength.com/

[112] K. Gjøsteen, "The Norwegian Internet Voting Protocol," Third International Conference on E-Voting and Identity (VoteID '11). Talinn, Estonia: Springer LNCS, Vol. 7187, 2012, pp. 1–18.

[113] J. Glatthard, "SWI swissinfo.ch - "Deutliches Ja in der Abstimmung zum Covid-Gesetz"," Nov. 2021, https://www.swissinfo.ch/ger/wirtschaft/resultat-covid-gesetz-zertifikat/47141410.

[114] O. Goldreich, "A uniform-complexity treatment of encryption and zero-knowledge," Journal of Cryptology, Vol. 6, No. 1, pp. 21–53, 1993. [Online]. Available: https://doi.org/10.1007/BF02620230

[115] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," 17th Annual ACM Symposium on Theory of Computing (STOC '85), Providence, Rhode Island, USA, 1985, p. 291–304.

[116] S. Goldwasser and S. Micali, "Probabilistic encryption amp; how to play mental poker keeping secret all partial information," Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, ser. STOC '82. New York, NY, USA: Association for Computing Machinery, 1982, p. 365–377. [Online]. Available: https://doi.org/10.1145/800070.802212

[117] Google, "Angular.io," https://angular.io/.

[118] Google, "Google Scholar," https://scholar.google.ch/.

[119] Google, "Material Design," https://material.io/.

[120] J. Groth, "Efficient maximal privacy in boardroom voting and anonymous broadcast," International Conference on Financial Cryptography. Springer, 2004, pp. 90–104.

[121] P. Grühnenfelder, "Neue Zürcher Zeitung: *Digitale Demokratie verlangt Pioniergeist*," September 2015, http://pvpf.ch/nzzpio.

[122] S. Guasch and P. Morillo, "How to challenge and cast your e-vote," Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 130–145, https://doi.org/10.1007/978-3-662-54970-4_8.

[123] S. Guasch Castelló, "Individual Verifiability in Electronic Voting," Ph.D. dissertation, Universitat Politècnica de Catalunya, Feb. 2016.

[124] R. Haenni, E. Dubuis, R. E. Koenig, and P. Locher, "CHVote: Sixteen Best Practices and Lessons Learned," Electronic Voting, R. Krimmer, M. Volkamer, B. Beckert, R. Küsters, O. Kulyk, D. Duenas-Cid, and M. Solvak, Eds. Cham: Springer International Publishing, 2020, pp. 95–111.

[125] R. Haenni, E. Dubuis, and U. Ultes-Nitsche, "Research on E-Voting Technologies: A Survey," 2008.

[126] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "CHVote System Specification 3.2," IACR, Vol. 2017, p. 325, 2017.

[127] R. Haenni and P. Locher, "Performance of Shuffling: Taking it to the Limits," International Conference on Financial Cryptography and Data Security. Springer, 2020, pp. 369–385.

[128] R. Haenni, P. Locher, and N. Gailly, "Improving the Performance of Cryptographic Voting Protocols," International Conference on Financial Cryptography and Data Security. Springer, 2019, pp. 272–288.

[129] R. Haenni, P. Locher, and N. Gailly, "Improving the Performance of Cryptographic Voting Protocols," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 11599 LNCS. Springer, 2020, pp. 272–288.

[130] R. Haenni, P. Locher, R. Koenig, and E. Dubuis, "Pseudo-code algorithms for verifiable re-encryption mix-nets," International Conference on Financial Cryptography and Data Security. Springer, 2017, pp. 370–384.

[131] T. Haines, R. Gore, and B. Sharma, "Did You Mix Me? Formally Verifying Verifiable Mix Nets in Electronic Voting," Cryptology ePrint Archive, Report 2020/1114, 2020, https://eprint.iacr.org/2020/1114.

[132] T. Haines and M. Johannes, "How Not To VoteAgain: Pitfalls of Scalable Coercion-Resistant E-Voting," Cryptology ePrint Archive, Report 2020/1406, 2020. [Online]. Available: https://eprint.iacr.org/2020/1406

[133] T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How Not To Prove Your Election Outcome," 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 644–660.

[134] T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How Not To Prove Your Election Outcome," 2020 IEEE Symposium on Security and Privacy (SP '20), San Francisco, CA, USA, 2020, pp. 644–660.

[135] T. Haines and J. Muller, "SoK: Techniques for Verifiable Mix Nets," 33rd IEEE Computer Security Foundations Symposium (CSF '20), Virtual Conference, 2020, pp. 49–64.

[136] H. Halpin, "A critique of immunity passports and w3c decentralized identifiers," https://arxiv.org/pdf/2012.00136.pdf, November 2020.

[137] F. Hao and P. Y. A. Ryan, Real-World Electronic Voting: Design, Analysis and Deployment, September 2020.

[138] F. Hao, P. Y. Ryan, and P. Zieliński, "Anonymous voting by two-round public discussion," IET Information Security, Vol. 4, No. 2, pp. 62–67, 2010.

[139] M. Hapsara, A. Imran, and T. Turner, "*E-Voting in Developing Countries*," Electronic Voting, R. Krimmer, M. Volkamer, J. Barrat, J. Benaloh, N. Goodman, P. Y. A. Ryan, and V. Teague, Eds. Cham: Springer International Publishing, 2017, pp. 36–55.

[140] D. Hardman, "Didcomm messaging," https://identity.foundation/didcomm-messaging/spec/.

[141] P. Hayes, "Benaloh Challenge," https://crates.io/crates/benaloh-challenge.

[142] C. Hébant, D. H. Phan, and D. Pointcheval, "Linearly-homomorphic signatures and scalable mix-nets," IACR International Conference on Public-Key Cryptography. Springer, 2020, pp. 597–627.

[143] S. Heiberg, T. Martens, P. Vinkel, and J. Willemson, "Improving the verifiability of the estonian internet voting scheme," International Joint Conference on Electronic Voting. Springer, 2016, pp. 92–107.

[144] S. Heiberg, T. Martens, P. Vinkel, and J. Willemson, "Improving the verifiability of the estonian internet voting scheme," International Joint Conference on Electronic Voting (E-Vote-ID 2016), R. Krimmer, M. Volkamer, J. Barrat, J. Benaloh, N. Goodman, P. Y. A. Ryan, and V. Teague, Eds. Springer LNCS, Vol. 10141, 2017, pp. 92–107.

[145] S. Heiberg and J. Willemson, "Verifiable internet voting in estonia," 2014 6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014 - IEEE Proceedings, 01 2015, doi: 10.1109/EVOTE.2014.7001135.

[146] M. Hirt, "Receipt-free *K*-out-of-*L* voting based on elgamal encryption," Towards Trustworthy Elections, New Directions in Electronic Voting, ser. Lecture Notes in Computer Science, D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutylowski, and B. Adida, Eds., Vol. 6000. Springer, 2010, pp. 64–82. [Online]. Available: https://doi.org/10.1007/978-3-642-12980-3_3

[147] M. Hirt, "Receipt-free K-out-of-L voting based on ElGamal encryption," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 6000 LNCS, 2010, pp. 64–82.

[148] A. Hofmann, "Security analysis and improvements of a blockchain-based remote electronic voting system," 2020, Master Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[149] M. Hooghe, "Trust and Elections," The Oxford Handbook of Social and Political Trust, 01 2017, pp. 617–631.

[150] M. Howard and S. Lipner, The Security Development Lifecycle. Microsoft Press Redmond, 2006, Vol. 8.

[151] Hyperledger, "*Fabric Wiki*," http://bcbev.ch/hlfdocs.

[152] IEEE, "IEEE Xplore," https://ieeexplore.ieee.org.

[153] A. E. Inc, "Trust boundary," https://patterns.arcitura.com/cloud-computing-patterns/basics/roles-and-boundaries/trust_boundary.

[154] IPFS GitHub contributors, "How IPFS works | IPFS Docs," 2022, https://docs.ipfs.tech/concepts/how-ipfs-works/.

[155] IPFS GitHub contributors, "IPFS Documentation | IPFS Docs," 2022, https://docs.ipfs.tech/.

[156] L. James, "Phishing exposed," https://dl.acm.org/doi/book/10.5555/1121597#secAbs, 2005.

[157] G. Jöhri, "Veraltete Software: Sicherheitslücken bei Erfassung von Abstimmungs-Ergebnissen," 2020, https://www.srf.ch/news/schweiz/veraltete-software-sicherheitsluecken-bei-erfassung-von-abstimmungs-ergebnissen.

[158] H. Jonker, S. Mauw, and J. Pang, "*Privacy and Verifiability in Voting Systems*," Computer Science Review, pp. Vol. 10, pp. 1 – 30, November 2013.

[159] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," Towards Trustworthy Elections, ser. Lecture Notes in Computer Science. Springer, 2010, Vol. 6000, pp. 37–63.

[160] A. Juels, D. Catalano, and M. Jakobsson, Coercion-Resistant Electronic Elections. Springer Berlin Heidelberg, 2010, pp. pp. 37–63.

[161] Kantonales Wahlbüro Aargau, "*Wahlen und Abstimmungen, Checkliste Allgemeine Arbeiten (Rahmenorganisation)*," Regierungsgebäude, Aarau, November 2017.

[162] F. Karayumak, M. M. Olembo, M. Kauer, and M. Volkamer, "Usability analysis of helios-an open source verifiable remote electronic voting system." EVT/WOTE, Vol. 11, No. 5, 2011.

[163] A. Kate, Y. Huang, and I. Goldberg, "Distributed key generation in the wild," IACR Cryptology ePrint Arch., Vol. 2012, p. 377, 2012.

[164] J. Katz and Y. Lindell, Introduction to Modern Cryptography (Chapman & HallCrc Cryptography and Network Security Series), 3rd ed. Chapman & HallCRC, 2007.

[165] A. Kerckhoff, "La Cryptographie Militaire - Première Partie," Journal des sciences militaires, Vol. IX, pp. pp. 5–38, 1 1883.

[166] A. Kerckhoff, "La Cryptographie Militaire - Seconde Partie," Journal des sciences militaires, Vol. IX, pp. pp. 161–191, 2 1883.

[167] C. F. Kerry and P. D. Gallagher, "Digital Signature Standard (DSS)," NIST, Tech. Rep., 2013. [Online]. Available: http://dx.doi.org/10.6028/NIST.FIPS.186-4

[168] KEYENCE Corporation, "What is a QR code?" 2023, https://www.keyence.com/ss/products/auto_id/barcode_lecture/basic_2d/qr/.

[169] Keystone-SDA, "SWI swissinfo.ch - Untersuchung wegen Wahlfälschung und Stimmenfang in Pruntrut JU," Dec. 2012, https://www.swissinfo.ch/ger/untersuchung-wegen-wahlfaelschung-und-stimmenfang-in-pruntrut-ju/34129884.

[170] Keystone-SDA, "SWI swissinfo.ch - Wallis leitet nach Wahlbetrug Administrativuntersuchung ein," Sep. 2018, https://www.swissinfo.ch/ger/wallis-leitet-nach-wahlbetrug-administrativuntersuchung-ein/44414482.

[171] A. Kiayias and M. Yung, "Self-tallying Elections and Perfect Ballot Secrecy," International Workshop on Public Key Cryptography (PKC 2002). Springer LNCS, Vol. 2274, 2002, pp. 141–158.

[172] C. Killer, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, and B. Stiller, "Provotumn: Decentralized, mix-net-based, and receipt-free voting system," 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2022, pp. 1–9.

[173] C. Killer, M. Knecht, C. Müller, B. Rodrigues, E. J. Scheid, M. Franco, and B. Stiller, "Æternum: A Decentralized Voting System with Unconditional Privacy," IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2021). Darlinghurst, Australia: IEEE, 5 2021, pp. 1–8.

[174] C. Killer and M. Limacher, "Digital Exposure of Traditional Swiss Voting Channels," Bern, Switzerland, oct 2019, https://www.youtube.com/watch?v=HQbbExpuaz8&t=761s.

[175] C. Killer, B. Rodrigues, R. Matile, E. Scheid, and B. Stiller, "*Design and Implementation of Cast-as-Intended Verifiability for a Blockchain-based Voting System*," Proceedings of the 2020 ACM Symposium on Applied Computing, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, *to be published.*

[176] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, and B. Stiller, "From Centralized to Decentralized Remote Electronic Voting," To appear, ser. To appear. Cham, Switzerland: To appear, To appear 2021, pp. 1–38.

[177] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, and B. Stiller, "From Centralized to Decentralized Remote Electronic Voting," Blockchains — A Handbook on Fundamentals, Platforms, and Applications, M. C. Sushmita Ruj, Salil Kanhere, Ed. Springer, August 2022, pp. 1–38.

[178] C. Killer, B. Rodrigues, E. J. Scheid, M. F. Franco, and B. Stiller, "Blockchain-based voting considered harmful?" IEEE Transactions on Network and Service Management, pp. 1–1, 2022.

[179] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, and B. Stiller, "Provotum: A Blockchain-Based and End-to-End Verifiable Remote Electronic Voting System," IEEE 45th Conference on Local Computer Networks (LCN). Sydney, Australia: IEEE, 2020.

[180] C. Killer and B. Stiller, "A Flow Analysis of Today's Swiss Postal Voting Process and a Respective Security Scrutiny," Department of Informatics IfI, University of Zurich, IfI Technical Report No. 2019-02, April 2019.

[181] C. Killer and B. Stiller, "The source code of the e-voting system is problematic, and that's not just about security (interview with Christian Killer, Burkhard Stiller)," p. 1, February 2019. [Online]. Available: https://doi.org/10.5167/uzh-185220

[182] C. Killer, L. Thorbecke, B. Rodrigues, E. Scheid, M. Franco, and B. Stiller, "Proverum: A Hybrid Public Verifiability and Decentralized Identity Management," Computing Research Repository (CoRR) in arXiv, 2020. [Online]. Available: https://arxiv.org/abs/2008.09841

[183] L. Kohnfelder and P. Garg, "The threats to our products," Microsoft Interface, Microsoft Corporation, Vol. 33, 1999.

[184] M.-J. Kolly, "Neue Zürcher Zeitung: *Der Quellcode des E-Voting-Systems ist problematisch, und das hat nicht nur mit Sicherheit zu tun*," March 2019, http://pvpf.ch/nzz.

[185] S. Kremer, M. Ryan, and B. Smyth, "Election verifiability in electronic voting protocols," ESORICS'10, 15th European Symposium on Research in Computer Security, Vol. 6345 LNCS, pp. 389–404, 2010.

[186] R. Krimmer, "*World Map of Electronic Voting*," http://www.e-voting.cc/en/it-elections/world-map/.

[187] R. Krimmer, *A Structure for New Voting Technologies: What They Are, How They Are Used and Why*. Cham: Springer International Publishing, 2019, pp. 421–426. [Online]. Available: https://doi.org/10.1007/978-3-030-06234-7_39

[188] R. Krimmer, S. Triessnig, and M. Volkamer, "*The Development of Remote E-Voting Around the World: A Review of Roads and Directions*," First International Joint Conference on Electronic Voting and Identity (E-VOTE ID 2007). Bochum, Germany, October 2008, pp. 1–15.

[189] R. Krimmer, M. Volkamer, and D. Duenas-Cid, "*E-Voting – An Overview of the Development in the Past 15 Years and Current Discussions*," Electronic Voting, R. Krimmer, M. Volkamer, V. Cortier, B. Beckert, R. Küsters, U. Serdült, and D. Duenas-Cid, Eds. Cham: Springer International Publishing, 2019, pp. 1–13.

[190] N. Kübler, "Design and implementation of a verifiable remote postal voting system," 2022, Master Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[191] N. Kübler, "Survey and Comparison of Blockchain-based Voting Systems," February 2022, Independent Study, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[192] R. Kuenzi, "SWI swissinfo.ch - "Genfer Fall ist nur die Spitze des Eisbergs"," May 2019, https://www.swissinfo.ch/ger/direktedemokratie/wahlbetrug-schweiz-direkte-demokratie/44959280.

[193] R. Kuenzi, "SWI swissinfo.ch - "Aus der Apfelrepublik Thurgau ist eine Bananenrepublik geworden"," Apr. 2020, https://www.swissinfo.ch/ger/direktedemokratie/wahlmanipulation_-aus-der-apfelrepublik-thurgau-ist-eine-bananenrepublik-geworden-/45718620.

[194] M. Kuhr, "10 Gründe für Progressive-Web-Apps," https://www.new-communication.de/neues/detail/10-gruende-fuer-progressive-web-apps, June 27. 2019.

[195] O. Kulyk, J. Henzel, K. Renaud, and M. Volkamer, "Comparing "Challenge-Based" and "Code-Based" Internet Voting Verification Implementations," Human-Computer Interaction – INTERACT 2019. Cham: Springer International Publishing, 2019, pp. 519–538, https://doi.org/10.1007/978-3-030-29381-9_32.

[196] O. Kulyk, V. Teague, and M. Volkamer, "Extending helios towards private eligibility verifiability," International Conference on E-Voting and Identity. Springer, 2015, pp. 57–73.

[197] T. Kurt, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein, "Protecting accounts from credential stuffing with password breach alerting," https://dl.acm.org/doi/10.1145/1102120.1102168, 2019.

[198] R. Küsters, T. Truderung, and A. Vogt, "Verifiability, privacy, and coercion-resistance: New insights from a case study," 2011 IEEE Symposium on Security and Privacy, 2011, pp. 538–553, doi: 10.1109/SP.2011.21.

[199] R. Küsters, T. Truderung, and A. Vogt, "Clash attacks on the verifiability of e-voting systems," 2012 IEEE Symposium on Security and Privacy, 2012, pp. 395–409, doi: 10.1109/SP.2012.32.

[200] R. Küsters and J. Müller, "Cryptographic security analysis of e-voting systems: Achievements, misconceptions, and limitations," International Joint Conference on Electronic Voting. Springer, 2017, pp. 21–41, https://doi.org/10.1007/978-3-319-68687-5_2.

[201] Legion of the Bouncy Castle Inc, "bouncycastle.org," 2022, https://www.bouncycastle.org/.

[202] M. Levi and L. Stoker, "Political Trust and Trustworthiness," Annual Review of Political Science, Vol. 3, No. 1, pp. 475–507, 2000. [Online]. Available: https://doi.org/10.1146/annurev.polisci.3.1.475

[203] P. Locher, "Unconditional Privacy in Remote Electronic Voting Theory and Practice," Ph.D. dissertation, University of Fribourg, 2016.

[204] P. Locher and R. Haenni, "Verifiable Internet Elections with Everlasting Privacy and Minimal Trust," Lecture Notes in Computer Science, Vol. 9269, pp. 74–91, 2015.

[205] P. Locher and R. Haenni, "Receipt-free remote electronic elections with everlasting privacy," Annales des Telecommunications/Annals of Telecommunications, 2016.

[206] P. Locher, R. Haenni, and R. Koenig, "Coercion-Resistant Internet Voting with Everlasting Privacy," Financial Cryptography and Data Security, Vol. 9604, 02 2016, pp. 161–175.

[207] P. Locher, R. Haenni, and R. E. Koenig, "*Coercion-Resistant Internet Voting with Everlasting Privacy*," Financial Cryptography and Data Security. Springer Berlin Heidelberg, August 2016, pp. pp. 161–175.

[208] S. Z. Lowry and P. L. Vora, "Desirable Properties of Voting Systems," NIST E2E Workshop, 2009. [Online]. Available: https://www.nist.gov/publications/desirable-properties-voting-systems

[209] S. Luechinger, M. Rosinger, and A. Stutzer, "*The Impact of Postal Voting on Participation: Evidence for Switzerland*," Swiss Political Science Review, pp. 167–202, January 2007.

[210] Z. A. Lux, D. Thatmann, S. Zickau, and F. Beierle, "Distributed-ledger-based authentication with decentralized identifiers and verifiable credentials," https://arxiv.org/pdf/2006.04754.pdf, June 2020.

[211] F. Maddaloni, "Voting Verification Mechanism for a Distributed Ledger based Remote Electronic Voting System," 2021, Master Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[212] K. Marky, O. Kulyk, K. Renaud, and M. Volkamer, What Did I Really Vote For? On the Usability of Verifiable E-Voting Schemes. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3173574.3173750

[213] Material UI SAS, "MUI: The React component library you always wanted," https://mui.com/.

[214] R. Matile and C. Killer, "*Privacy, Verifiability, and Auditability in Blockchain-based E-Voting*," 2018, Master Project, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland, 2020. [Online]. Available: http://bcbev.ch/pv

[215] P. McCorry, S. F. Shahandashti, and F. Hao, "A Smart Contract for Boardroom Voting with Maximum Voter Privacy," Lecture Notes in Computer Science (LNCS), 2017, Vol. 10322, pp. 357–375.

[216] G. McGraw, "Software security," IEEE Security Privacy, Vol. 2, No. 2, pp. 80–83, 2004.

[217] E. McMurtry, X. Boyen, C. Culnane, K. Gjøsteen, T. Haines, and V. Teague, "Towards Verifiable Remote Voting with Paper Assurance," 2021. [Online]. Available: https://arxiv.org/abs/2111.04210

[218] Meta Platforms, Inc., "React - introducing hooks," https://reactjs.org/docs/hooks-intro.html.

[219] Meta Platforms, Inc., "React - A JavaScript library for building user interfaces," 2022, https://reactjs.org/.

[220] Microsoft, "TypeScript: JavaScript With Syntax For Types." 2022, https://www.typescriptlang.org/.

[221] M. B. Mikhail J. Atallah, Ed., Algorithms and Theory of Computation Handbook. CRC Press, 2009.

[222] T. Milic, M. McArdle, and U. Serdült, "*Haltungen und Bedürfnisse der Schweizer Bevölkerung zu E-Voting = Attitudes of Swiss Citizens Towards the Generalisation of E-Voting*," Aarau, Studienbericht, September 2016.

[223] V. Mohan and L. B. Othmane, "Secdevops: Is it a Marketing Buzzword? Mapping Research on Security in Devops," 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE, 2016, pp. 542–547.

[224] X. Monnat and S. Oswald, "The Challenges of Enabling Public Scrutiny," Electronic Voting. E-Vote-ID 2021., pp. 289–303, 2021.

[225] T. Moran and M. Naor, "Receipt-free Universally-verifiable Voting with Everlasting Privacy," Advances in Cryptology (CRYPTO '06). Springer LNCS, Vol. 4117, 2006, pp. 373–392.

[226] C. Mueller, "Github: Aeternum - UP Voting System." [Online]. Available: https://github.com/provotum/up-voting-system

[227] H. Myrbakken and R. Colomo-Palacios, "DevSecOps: A Multivocal Literature Review," Software Process Improvement and Capability Determination, A. Mas, A. Mesquida, R. V. O'Connor, T. Rout, and A. Dorling, Eds. Cham: Springer International Publishing, 2017, pp. 17–29.

[228] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," https://dl.acm.org/doi/10.1145/1102120.1102168, November 2005.

[229] National Institute of Standards and Technology (NIST), "Post-Quantum Cryptography." [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[230] A. C. Neff, "Practical high certainty intent verification for encrypted votes," 2004.

[231] S. Neumann, C. Feier, M. Volkamer, and R. Koenig, "Towards a practical jcj/civitas implementation," 2013.

[232] nVotes, "nVotes - Secure Online Voting Software." [Online]. Available: https://nvotes.com

[233] T. Okamoto and K. Ohta, "Divertible zero knowledge interactive proofs and commutative random self-reducibility," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 434 LNCS. Springer Verlag, 1990, pp. 133–149.

[234] OQS (Open Quantum Safe) Project, "TLS." [Online]. Available: https://openquantumsafe.org/applications/tls/

[235] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," Advances in Cryptology (EUROCRYPT '99), J. Stern, Ed. Prague, Czech Repbulic: Springer LNCS, Vol. 1592, 1999, pp. 223–238.

[236] Pammet, H. Jon and Goodman, Nicole, "*Consultation and Evaluation Practices in the Implementation in the Implementation of Internet Voting in Canada and Europe,*" Research Study, November, 2013.

[237] Parity Technologies. Substrate: The Platform for Blockchain Innovators. [Online]. Available: https://github.com/paritytech/substrate

[238] Parity Technologies. Substrate v3.0/0.9 – Apollo 14. [Online]. Available: https://github.com/paritytech/substrate/releases/tag/v3.0.0

[239] C. Park, K. Itoh, and K. Kurosawa, "Efficient Anonymous Channel and All/Nothing Election Scheme," Workshop on the Theory and Application of of Cryptographic Techniques, 1993, pp. 248–259.

[240] S. Park, M. A. Specter, N. Narula, and R. L. Rivest, "Going from Bad to Worse: From Internet Voting to Blockchain Voting," J. Cybersecur., Vol. 7, No. 1, 2021. [Online]. Available: https://doi.org/10.1093/cybsec/tyaa025

[241] Parlamentarische Verwaltungskontrolle (PVK), "*Elektronische Auszählung von Stimmen (E-Counting) Bericht der PVK zuhanden der Geschäftsprüfungs-kommission des Nationalrates,*" Februar 2017.

[242] Pauchard O. - Swissinfo, "*Tausende beim Zählen der Wahlzettel,*" http://pvpf.ch/swi, October 2003.

[243] T. P. Pedersen, "A threshold cryptosystem without a trusted party," EUROCRYPT, ser. Lecture Notes in Computer Science. Springer, 1991, Vol. 547, pp. 522–526.

[244] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," Lecture Notes in Computer Science, Vol. 576 LNCS, pp. 129–140, 1992.

[245] A. Preukschat and D. Reed, SELF-SOVEREIGN IDENTITY: Decentralized Digital Identity and Verifiable Credentials. O'Reilly Media, 2021.

[246] Proverum, "*GitHub,*" http://pvpf.ch/proverum.

[247] J. Puigalli and S. Guasch, "Privacy and Anonymity Management in Electronic Voting," Identity and Privacy Management, Vol. XI, No. 1, pp. 59–95, 2010.

[248] C. Raaflaub, "SWI swissinfo.ch - "Stimmvolk heisst Covid-19-Gesetz gut"," Jun. 2021, https://www.swissinfo.ch/ger/abstimmung-schweiz-13--juni-2021_-covid-19-gesetz_resultat/46697724.

[249] D. Reed, M. Sporny, D. Longley, C. Allen, and R. G. M. Sabadello, "JSON-LD 1.1 A JSON-based Serialization for Linked Data W3C Recommendation 16 July 2020 - 1.7 Syntax Tokens and Keywords," https://www.w3.org/TR/json-ld11/#syntax-tokens-and-keywords.

[250] D. Reed, M. Sporny, D. Longley, C. Allen, and R. G. M. Sabadello, "Decentralized identifiers (dids) v1.0," https://www.w3.org/TR/did-core/, November 2021.

[251] A. Regenscheid and N. Hastings, "*A Threat Analysis on VOCAVA Voting Systems*," US Department of Commerce, NIST, Threat Analysis, December 2008.

[252] A. Regenscheid and N. Hastings, "*Guide for Conducting Risk Assessments*," US Department of Commerce, NIST, NIST Special Publication 800-30, September 2012.

[253] M. Richards and N. Ford, Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, Inc., 2020.

[254] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, Vol. 21, No. 2, pp. pp. 120–126, 2 1978.

[255] R. L. Rivest, "On the Notion of 'Software Independence' in Voting Systems," Philosophical Transactions of the Royal Society Mathematical, Physical and Engineering Sciences, Vol. 366, No. 1881, pp. 3759–3767, 2008.

[256] M. D. Robbins, "Myths, values and digital transformation: The exceptional case of estonia," 04 2018.

[257] P. Rogaway, "The Moral Character of Cryptographic Work," Cryptology ePrint Archive, Report 2015/1162. [Online]. Available: https://eprint.iacr.org/2015/1162.pdf

[258] D. Roth, R. Anderson, and S. Luttin, "Overview of ASP.NET Core | Microsoft Docs," 2022, https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0.

[259] M. D. Ryan and B. Smyth, "Applied Pi Calculus," Cryptology and Information Security Series, Vol. 5, No. 2011, pp. 112–142, 2011.

[260] P. Y. Ryan, P. B. Rønne, and V. Iovino, "Selene: Voting with Transparent Verifiability and Coercion-Mitigation." Christ Church, Barbados: Springer LNCS, Vol. 9604, 2016, pp. 176–192.

[261] M. Sabadello, "A Universal Resolver for Self-Sovereign Identifiers," https://medium.com/decentralized-identity/a-universal-resolver-for-self-sovereign-identifiers-48e6b4a5cc3c, November 2017.

[262] M. Sabadello, K. Den Hartog, C. Lundkvist, C. Franz, A. Elias, A. Hughes, J. Jordan, and D. Zagidulin, "Introduction to DID Auth," https://github.com/WebOfTrustInfo/rwot6-santabarbara/blob/master/final-documents/did-auth.md.

[263] K. Sako and J. Kilian, "*Receipt-Free Mix-Type Voting Scheme*," Advances in Cryptology, (EUROCRYPT '95). Springer, Berlin Heidelberg, May 1995, pp. pp. 393–403.

[264] J. Sanders and D. Patterson, "Facebook data privacy scandal: A cheat sheet," https://www.techrepublic.com/article/facebook-data-privacy-scandal-a-cheat-sheet/.

[265] M. Sandmeier, "Stadtschreiber und Leiter Stadtkanzlei Baden, February 22, 2019," Stadtkanzlei, Baden, Personal Conversation.

[266] A. Sax, "Leiterin Wahlen und Abstimmungen, February 20, 2019," Staatskanzlei, Generalsekretariat, Kanton Aargau, Regierungsgebäude, Aarau, Personal Conversation.

[267] E. J. Scheid, P. Kiechl, M. Franco, B. Rodrigues, C. Killer, and B. Stiller, "Security and Standardization of a Notary-based Blockchain Interoperability API," IEEE International Conference on Blockchain Computing and Applications (BCCA 2021), Tartu, Estonia, November 2021, pp. 1–7.

[268] E. J. Scheid, B. Rodrigues, C. Killer, M. Franco, S. Rafati, and B. Stiller, "Blockchains and Distributed Ledgers Uncovered: Clarifications, Achievements, and Open Issues," Advancing Research in Information and Communication Technology, ser. IFIP AICT Festschrifts. Cham, Switzerland: Springer, August 2021, Vol. 600, pp. 289–317.

[269] C. Schnorr, "Efficient signature generation by smart cards," Journal of Cryptology, Vol. 4, No. 3, pp. 161–174, 1991.

[270] V. D. Schutter, "Deep dive into the security of Progressive Web Apps," https://blog.nviso.eu/2020/01/16/deep-dive-into-the-security-of-progressive-web-apps/, January 16. 2020.

[271] Scytl Secure Electronic Voting, "Swiss Online Voting Protocol," https://www.post.ch/-/media/post/evoting/dokumente/swiss-post-online-voting-protocol.pdf?la=en, November 2017.

[272] H. N. Security, "How secure are open source libraries?" https://www.helpnetsecurity.com/2020/05/21/secure-open-source-libraries/, May 2020.

[273] U. Serdült, E. Dubuis, and A. Glaser, "*Elektronischer versus brieflicher Stimmkanal im Vergleich. Überprüfbarkeit, Sicherheit und Qualität der Stimmabgabe*," Jusletter IT, September, 2017.

[274] A. Shamir, "How to share a secret," Commun. ACM, Vol. 22, No. 11, p. 612–613, Nov. 1979.

[275] A. Shostack, Threat Modeling: Designing for Security. John Wiley & Sons, Inc., 2014.

[276] Silvan Meile - Tagblatt, "*Wahlbetrug im Thurgau – Gericht spricht Ex-Stadtschreiber von Frauenfeld schuldig*," https://pvpf.ch/frauenfeld, July 2021.

[277] simibac, "Looking for resources about srd #356," https://github.com/uport-project/veramo/discussions/356.

[278] N. P. Smart, *Cryptography Made Simple*. Springer International Publishing, 2016.

[279] Smartmatic, "Online Voting - Frequently Asked Questions," 05 2020, https://www.smartmatic.com/fileadmin/user_upload/Smartmatic_TIVI_FAQ_May2020.pdf.

[280] R. Smith, "*Implications of Changes to Voting Channels in Australia*," Research Report comm. by the Electoral Regulation Research Network, December, 2018.

[281] W. D. Smith, "Cryptography meets voting," Compute, Vol. 10, p. 64, 2005. [Online]. Available: http://www.hit.bme.hu/~buttyan/courses/BMEVIHI5316/Smith.Crypto%5Fmeets%5Fvoting.pdf

[282] B. Smyth, "A foundation for secret, verifiable elections," Cryptology ePrint Archive, Paper 2018/225, 2018, https://eprint.iacr.org/2018/225. [Online]. Available: https://eprint.iacr.org/2018/225

[283] B. Smyth, S. Frink, and M. R. Clarkson, "Election verifiability: Cryptographic definitions and an analysis of helios, helios-c, and jcj," Cryptology ePrint Archive, Report 2015/233, 2015, https://eprint.iacr.org/2015/233.

[284] Solidity Team, "Solidity Programming Language," 2022, https://soliditylang.org/.

[285] D. M. Sommer, M. Schneider, J. Gut, and S. Capkun, "*Cyber-Risks in Paper Voting*," CoRR, 2019. [Online]. Available: http://arxiv.org/abs/1906.07532

[286] M. A. Specter, J. Koppel, and D. Weitzner, "The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, The First Internet Voting Application used in U.S. Federal Elections," USENIX 29th Security Symposium, pp. 1535–1552, 2020.

[287] M. Sporny, D. Longley, and D. Chadwick, "Verifiable credentials data model 1.0," https://www.w3.org/TR/vc-data-model/, November 2019.

[288] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin, and N. Lindström, "Json-ld 1.1 - a json-based serialization for linked data," https://www.w3.org/TR/json-ld11/, July 2020.

[289] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security analysis of the Estonian Internet voting system," Proceedings of the 21st ACM Conference on Computer and Communications Security. ACM, Nov. 2014.

[290] O. Spycher, R. Koenig, R. Haenni, and M. Schläpfer, "A new approach towards coercion-resistant remote e-voting in linear time," International Conference on Financial Cryptography and Data Security. Springer, 2011, pp. 182–189.

[291] Staatssekretariat für Wirtschaft SECO, Schweiz, "*Nationale E-Government Studie 2019*," http://pvpf.ch/egov19, March 2019.

[292] Stadt Zürich, "Zuzug," https://pvpf.ch/zuzug.

[293] W. Stallings, Network Security Essentials: Applications and Standards, 6th Edition. Pearson Education, 2017.

[294] Statista, "Anteil der Besitzer von Smartphones in der Schweiz von 2017 bis 2020," https://bcbev.ch/smartphone.

[295] R. Staubli. Provotum CI/CD Source Code. [Online]. Available: https://github.com/provotum/provotum-ci-cd

[296] R. Staubli, "Secure Deployment and Configuration Management for a Decentralized Remote Electronic Voting System," 2021, Master Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland.

[297] O. Steele and M. Sporny, "DID Specification Registries," https://www.w3.org/TR/did-spec-registries/.

[298] O. Steele and M. Sporny, "DID Specification Registries - DID Methods," https://www.w3.org/TR/did-spec-registries/#did-methods.

[299] K. Stine, R. Kissel, W. C. Barker, J. Fahlsing, and J. Gulick, "*Volume I: Guide for Mapping Types of Information and Information Systems to Security Categories*," US Department of Commerce, NIST, NIST Special Publication 800-60, Vol. I, Rev. 1, August 2008.

[300] K. Stine, R. Kissel, W. C. Barker, A. Lee, and J. Fahlsing, "*Volume II: Appendices to Guide for Mapping Types of Information and Information Systems to Security Categories*," US Department of Commerce, NIST, NIST Special Publication 800-60, Vol. II, Rev. 1, August 2008.

[301] Storybook, "Storybook - build uis without the grunt work," https://storybook.js.org/.

[302] Swiss Post (Die Schweizerische Post) AG, "Security by Design in Swiss Post's New e-Voting System, v1.00," 10.06.2021. [Online]. Available: https://bcbev.ch/securitybydesign

[303] Swiss Post (Die Schweizerische Post) AG, "SwissPost Voting System Architecture Document, v0.9.1," 2021. [Online]. Available: https://bcbev.ch/swisspostarch

[304] B. Terelius and D. Wikström, "Proofs of restricted shuffles," International Conference on Cryptology in Africa. Springer, 2010, pp. 100–113.

[305] United Nations Human Rights Office of the High Commissioner, "*Human Rights and Elections: A Handbook on International Human Rights Standards on Elections*," Professional Training Series No. 2/Rev.1, new York and Geneva, 2021.

[306] United Nations (UN) General Assembly, "Universal Declaration of Human Rights," 1948.

[307] unknown, "Using privacy-preserving zkp credentials on the mattr platform," https://mattr.global/using-privacy-preserving-zkp-credentials-on-the-mattr-platform/, September 2020.

[308] B. van Spyk, "Vizestaatssekretär Kanton St. Gallen, February 27, 2019," Staatskanzlei, Recht und Legistik, Regierungsgebäude St. Gallen, Personal Conversation.

[309] S. Venugopalan, I. Homoliak, Z. Li, and P. Szalachowski, "BBB-voting: 1-out-of-k blockchain-based boardroom voting," 2021, _eprint: 2010.09112.

[310] Veramo, "Performant and modular APIs for Verifiable Data and SSI," https://veramo.io/.

[311] Veramo, "Veramo - jwtmessagehandler class," https://veramo.io/docs/api/did-jwt.jwtmessagehandler.

[312] Veramo Core Development, "Ethr-did library," https://github.com/uport-project/ethr-did.

[313] Veramo Core Development, "Ethr-did library," https://github.com/uport-project/ethr-did-registry.

[314] Verein eCH, "*eCH-Standards*," http://pvpf.ch/ech.

[315] Vocdoni, "The voice of digital voting," https://vocdoni.io.

[316] M. Volkamer, O. Spycher, and E. Dubuis, "*Measures to Establish Trust in Internet Voting*," ICEGOV 2011, Proceedings of the 5th International Conference on Theory and Practice of Electronic Governance (ICEGOV'11), Tallinn, Estonia, E. Estevez and M. Janssen, Eds. ACM, 2011, pp. 1–10.

[317] W3C Credentials Community Group, "Bbs+ signatures 2020," https://github.com/w3c-ccg/vc-di-bbs, Mai 2021.

[318] L. Wagner, "Shamir's secret sharing step-by-step," https://qvault.io/2020/08/18/very-basic-shamirs-secret-sharing/, August 2001.

[319] I. WalletConnect, "Walletconnect docs - mobile linking," https://veramo.io/docs/api/did-jwt.jwtmessagehandler.

[320] H. Weber, "Tagesanzeiger: *Unbekannte setzen Briefkasten mit Wahlcouverts in Brand*," October 2019, http://pvpf.ch/ta2019brand.

[321] S. G. Weber, R. Araujo, and J. Buchmann, "On coercion-resistant electronic elections with linear work," The Second International Conference on Availability, Reliability and Security (ARES'07). IEEE, 2007, pp. 908–916.

[322] D. Wikström, "Verificatum Mix-Net (VMN)," Sep. 2020. [Online]. Available: https://www.verificatum.org/html/product_vmn.html

[323] P. Windley and D. Reed, "*Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust*," The Sovrin Foundation, Whitepaper, January 2018.

[324] Wolfram Alpha LLC, "WolframAlpha," https://pvpf.ch/wolfram.

[325] Wolfram Alpha LLC, "WolframAlpha," https://pvpf.ch/wolframtally.

[326] D. Wong, Real-World Cryptography. Manning Publications Co., 2021.

[327] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks," Computing Research Repository (CoRR) in arXiv, 2019. [Online]. Available: http://arxiv.org/abs/1904.04098

[328] S. Yilmaz and I. Sertkaya, "Improving the individual verification of estonian internet voting scheme," 2020 International Conference on Information Security and Cryptology (ISCTURKEY), 2020, pp. 38–47, doi: 10.1109/ISCTURKEY51113.2020.9308016.

[329] S. Zhang, L. Wang, and H. Xiong, "Chaintegrity: blockchain-enabled large-scale e-voting system with robustness and universal verifiability," International Journal of Information Security, Vol. 19, No. 3, pp. 323–341, 06 2020. [Online]. Available: https://doi.org/10.1007/s10207-019-00465-8

All links provided above were last accessed on September 18, 2023.

A Publications

In the course of this PhD thesis, several scientific articles, book chapters, and tutorials were published. These publications are in the field of EV, cyber security andor BC and DL applications, and therefore in direct relation to this PhD thesis. Further, multiple Master's and Bachelor's Theses (as well as other theses) were supervised by the author of this thesis, whereas most of the results obtained directly contributed to the development of this PhD thesis.

In Table A.1 contributions are organized per chapter and publication type (*i.e.*, Full Papers, Short Papers, Journals, Book Chapters, Technical Reports (TR), Master Theses (MSC), Master Project Reports (MAP), Bachelor Theses (BSC) and Independent Studies (IS). These contributions were developed and supervised during the doctoral program duration. When these contributions were used directly in this PhD thesis, they are cited and listed in Table A.1.

Table A.1: List of Publications per Chapter

Chapter	Related Publications				Student Theses			
	Full Paper	Journal	Book Chapter	TR	MSc	MAP	BSc	IS
1. Introduction	[172, 173]	[178]	[177]					
2. Cryptography Foundations	[172]	[178]			[91, 148]	[92]		
3. Voting System Fundamentals		[178]	[177]					
4. Literature Review		[178]	[177]		[296]			[191]
5. Swiss RPV Process	[180]							
6. Decentralized RPV				[182]	[190]			
7. Decentralized REV	[172]	[178]			[12, 91, 211]			
8 Summary, Conclusions								

Publication Details
[12] Identity Management in a Decentralized Remote Electronic Voting System
[91] Mixnets in a Distributed Ledger Remote Electronic Voting System
[92] Design and Implementation of Blockchain-based E-Voting
[148] Security Analysis and Improvements of a Blockchain-based Remote Electronic Voting System
[179] Provotum: A Blockchain-Based and End-to-End Verifiable Remote Electronic Voting System
[182] Proverum: A Hybrid Public Verifiability and Decentralized Identity Management
[173] Æternum: A Decentralized Voting System with Unconditional Privacy
[190] Design and Implementation of a Verifiable Remote Postal Voting System
[191] Survey and Comparison of Blockchain-based Voting Systems
[172] ProvotuMN: Decentralized, Mix-Net-based, and Receipt-free Voting System
[177] From Centralized to Decentralized Remote Electronic Voting
[178] Blockchain-based Voting Considered Harmful?
[211] Voting Verification Mechanism for a Distributed Ledger based Remote Electronic Voting System
[296] Secure Deployment and Configuration Management for a Decentralized Remote Electronic Voting System

A.1 List of First Author Publications

2022

- *[Journal]* **C. Killer**, B. Rodrigues, E. J. Scheid, M. F. Franco, B. Stiller: **Blockchain-based Voting Considered Harmful?**; IEEE Transactions on Network and Service Management (TNSM), Vol. 19, No. 3, June 2022, pp. 3603-36.
- *[Full Paper]* **C. Killer**, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, B. Stiller: **ProvotuMN: Decentralized, Mix-Net-based, and Receipt-free Voting System**; IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2022), Virtually, May 2021.

2021

- *[Tutorial]* **C. Killer**, B. Rodrigues, E. J. Scheid, M. Franco, B. Stiller: **Blockchain-based Remote Electronic Voting from Theory to Practice**; IFIP/IEEE International Symposium on Integrated Network Management (IM 2021), Bordeaux, France, May 2021.
- *[Full Paper]* **C. Killer**, M. Knecht, C. Müller, B. Rodrigues, E. Scheid, M. Franco, B. Stiller: **Æternum: A Decentralized Voting System with Unconditional Privacy**; IEEE Interna-

tional Conference on Blockchain and Cryptocurrency (ICBC 2021), Virtually, Darlinghurst, Australia, May 2021, pp. 1–8.

2020

- *[Full Paper]* **C. Killer**, B. Rodrigues, R. Matile, E. J. Scheid, B. Stiller: **Design and Implementation of Cast-as-Intended Verifiability for a Blockchain-based Voting System**. The 35th ACM Symposium on Applied Computing (SAC). March 2020. Brno, Czech Republic
- *[Full Paper]* **C. Killer**, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Schetlin, B. Stiller: **Provotum: A Blockchain-Based and End-to-End Verifiable Remote Electronic Voting System**; IEEE 45th Conference on Local Computer Networks (LCN 2020), Sydney, Australia, November 2020, pp. 1-12.
- *[Tutorial]* **C. Killer**, B. Rodrigues, E. Scheid, M. Franco, B. Stiller: **Practical Introduction to Blockchain-based Remote Electronic Voting**; IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2020), May 2020, Toronto, Canada.
- *[Technical Report]* **C. Killer**, L. Thorbecke, B. Rodrigues, E. J. Scheid, M. Franco, B. Stiller: **Proverum: A Hybrid Public Verifiability for Decentralized Identity Management**; Technical Report No. 2020.03, Department of Informatics IfI, Universität Zürich UZH, Zürich, Switzerland, August 2020.

2019

- *[Workshop Paper]* **C. Killer**, B. Rodrigues, B. Stiller: **Threat Management Dashboard for a Blockchain Collaborative Defense**; IEEE GLOBECOM 2019 Workshop on Telecommunications and Blockchain. Waikoloa, HI, USA, December 2019.
- *[Short-paper]* **C. Killer**, B. Rodrigues, B. Stiller: **Security Management and Visualization in a Blockchain-based Collaborative Defense**; IEEE International Conference on Blockchain and Cryptocurrency (ICBC 19). Seoul, South Korea, May 2019.
- *[Full Paper]* **C. Killer**, B. Stiller: **The Swiss Postal Voting Process and its System and Security Analysis**; 2019 The International Conference for Electronic Voting (E-VOTE ID), Bregenz, Austria, October 2019.

A.2 List of Co-Authorship Publications

2022

- *[Journal]* E. J. Scheid, R. Hy, M. F. Franco, **C. Killer**, B. Stiller: **On the Employment of Machine Learning in the Blockchain Selection Process**; IEEE Transactions on Network and Service Management, Vol. 1, No. 1, December 2022, pp. 1–12.

- *[Full Paper]* J. von der Assen, M. F. Franco, **C. Killer**, E. J. Scheid, B. Stiller: **CoReTM: An Approach Enabling Cross-Functional Collaborative Threat Modeling**; IEEE International Conference on Cyber Security and Resilience, Rhodes, Greece, July 2022, pp. 1–8.
- *[Book Chapter]* B. Rodrigues, M. Franco, **C. Killer**, E. J. Scheid, B. Stiller: **On Trust, Blockchain, and Reputation Systems**; Handbook on Blockchain, Series on Optimization and Its Applications, Springer, Cham, Switzerland, No. 194, 2022, ISBN 978-3-031-07534-6, pp. 299–337.
- *[Technical Report]* J. von der Assen, M. F. Franco, **C. Killer**, E. J. Scheid, B. Stiller: **On Collaborative Threat Modeling**; UZH, Technical Report No. 2022.04, Department of Informatics IfI, Universität Zürich UZH, Zürich, Switzerland, April 2022.

2021

- *[Full Paper]* M. Franco, J. von der Assen, L. Boillat, **C. Killer**, B. Rodrigues, E. J. Scheid, L. Granville, B. Stiller: **SecGrid: A Visual System for the Analysis and ML-Based Classification of Cyberattack Traffic**; IEEE 46th Conference on Local Computer Networks (LCN 2021), Virtually, Edmonton, Canada, October 2021, pp. 1–8.
- *[Full Paper]* M. Franco, N. Berni, E. Scheid, B. Rodrigues, **C. Killer**, B. Stiller: **SaCI: a Blockchain -based Cyber Insurance Approach for the Deployment and Management of a Contract Coverage**; 18th International Conference on the Economics of Grids, Clouds, Systems and Services (GECON 2021), Virtually, September 2021, pp. 1–14.
- *[Journal]* M. Franco, B. Rodrigues, **C. Killer**, E. Scheid, A. De Carli, A. Gassmann, D. Schoenbaechler, B. Stiller: **WeTrace: a Privacy-preserving Tracing Approach**; KICKS and IEEE ComSoc, Journal of Communications and Networks (JCN), Special Issue on Communications and Networking Approaches for Combatting COVID-19, October 2021, pp. 1–16.
- *[Short Paper]* M. Franco, J. von der Assen, L. Boillat, **C. Killer**, B. Rodrigues, E. Scheid, L. Granville, B. Stiller, B. Rodrigues, B. Stiller: **DDoSGrid: a Platform for the Post-mortem Analysis and Visualization of DDoS Attacks**; IFIP Networking 2021, Virtually, Espoo, Finland, June 21-24 2021, pp. 1–3.
- *[Full Paper]* E. Scheid, A. Knecht, T. Strasser, **C. Killer**, M. Franco, B. Rodrigues, B. Stiller: **Edge2BC: a Practical Approach for Edge-to-Blockchain IoT Transactions**; IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2021), Virtually, Darlinghurst, Australia, May 2021, pp. 1–9.
- *[Short Paper]* R. Ribeiro, B. Rodrigues, **C. Killer**, L. Baumann, M. Franco, E. Scheid, B. Stiller: **ASIMOV: A Fully Passive WiFi Device Tracking**; IFIP Networking 2021, Virtually, Espoo, Finland, June 2021, pp. 1–3.

- *[Short Paper]* L. Mueller, B. Rodrigues, E. Scheid, M. Franco, **C. Killer**, B. Stiller: **LaFlector: A Privacy-Preserving LiDAR-Based Approach for Accurate Indoor Tracking**; IEEE 46th Conference on Local Computer Networks (LCN 2021), Virtually, Edmonton, Canada, October 2021, pp. 1–4.

- *[Book Chapter]* E. Scheid, B. Rodrigues, **C. Killer**, M. Franco, S. Rafati, B. Stiller: **Blockchains and Distributed Ledgers Uncovered: Clarifications, Achievements, and Open Issues**; Advancing Research in Information and Communication Technology, Springer, Cham, Switzerland, No. 1, August 2021, ISBN 978-3-030-81701-5, pp. 1–29.

- *[Full Paper]* E. J. Scheid, P. Kiechl, M. Franco, B. Rodrigues, **C. Killer**, B. Stiller: **Security and Standardization of a Notary-based Blockchain Interoperability API**; 3rd International Conference on Blockchain Computing and Applications (BCCA 2021), Tartu, Estonia, November 2021, pp. 1–7.

2020

- *[Journal]* B. Rodrigues, E. Scheid, **C. Killer**, M. Franco, B. Stiller: **Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks**; Journal of Network and Systems Management, Springer Nature, August, 2020, pp. 1-37.

- *[Full Paper]* M. Franco, B. Rodrigues, E. Scheid, A. Jacobs, **C. Killer**, L. Granville, B. Stiller: **SecBot: a Business-Driven Conversational Agent for Cybersecurity Planning and Management**; 16th International Conference on Network and Service Management (CNSM 2020), Mini-Conference, Izmir, Turkey, November 2020, pp. 1-7.

- *[White Paper]*. A. De Carli, M. Franco, A. Gassmann, **C. Killer**,, E. Scheid, D. Schoenbaechler, B. Stiller: **WeTrace: Privacy-preserving Mobile COVID-19 Tracing Approach and Application**; Technical Report, Zürich, Switzerland, April 2020.

B

Cryptographic Algorithms

The following algorithms are subalgorithms of the Shuffle proofs presented in Section 2.9.5 and based on [126].

GenerateShuffleProof

P knows p q g h

INPUTS: permutation ψ j_1 j_N Ψ_N public key pk, vote identifier id_{vote}
encryptions e e_1 e_N e_i a_i b_i g^{r_i} pk^{r_i} m
shuffled encryptions e e_1 e_N , re-encryption randoms r r_1 r_N $^{N}_{q}$

h GetIndependentGenerators N id_{vote} h h_1 h_N h_i $_p$ 1

c r GeneratePermutationCommitments ψ h

c c_1 c_N c_i $_p$ r r_1 r_N r_i $_q$

u GetChallenges N e e c pk u u_1 u_N u_i $_q$

u u_1 u_N u_i u_{j_i}

c r GenerateCommitmentChain u

c c_1 c_N c_i $_p$ r r_1 r_N r_i $_q$

R_o o U_o 1

for i 1 N **do**

w_i w_i $_r$ $_q$

R_i r_i u_i $R_{i\ 1}$ mod q R_i w_i w_i $R_{i\ 1}$ mod q

U_i u_i $U_{i\ 1}$ mod q U_i w_i $U_{i\ 1}$ mod q

t_i g^{R_i} h^{U_i} mod q

w_1 w_2 w_3 w_4 $_r$ $_q$

t_1 g^{w_1} mod p

t_2 g^{w_2} mod p

t_3 g^{w_3} $\prod_{i\ 1}^{N} h_i^{w_i}$ mod p

$t_{4,1}$ $t_{4,2}$ pk^{w_4} $\prod_{i\ 1}^{N} b_i^{w_i}$ mod p g^{w_4} $\prod_{i\ 1}^{N} a_i^{w_i}$ mod p

c hash **e e c c** pk t_1 t_2 t_3 $t_{4,1}$ $t_{4,2}$ t_i t_N mod q

v v_1 v_N v_N 1 $v_{i\ 1}$ u_i v_i mod q

r $\sum_{i\ 1}^{N} r_i$ mod q r $\sum_{i\ 1}^{N} r_i\ v_i$ mod q r $\sum_{i\ 1}^{N} r_i\ u_i$ mod q r $\sum_{i\ 1}^{N} r_i\ u_i$ mod q

s_1 w_1 c r mod q

s_2 w_2 c r mod q

s_3 w_3 c r mod q

s_4 w_4 c r mod q

s s_1 s_N s_i w_i c r_i mod q

s s_1 s_N s_i w_i c u_i mod q

OUTPUT: π c s_1 s_2 s_3 s_4 **s s c c**

Figure B.1: Shuffle Proof Generation

VerifyShuffleProof

known: p q g h

inputs:

shuffle proof π c s_1 s_2 s_3 s_4 $\mathbf{s}$ $\mathbf{s}$ $\mathbf{c}$ $\mathbf{c}$

encryptions $\mathbf{e}$ e_1 e_N e_i a_i b_i g^{r_i} pk^{r_i} m

shuffled encryptions $\mathbf{e}$ e_1 e_N

public key pk

vote identifier id_{vote}

h GetIndependentGenerators N id_{vote} h h_1 h_N h_i p 1

u GetChallenges N e e c pk u u_1 u_N u_i q

c_0 h

N N

c c_i h_i mod p

i 1 i 1

N

u u_i mod q

i 1

c c_N h^u mod p

N

c $c_i^{u_i}$ mod p

i 1

N N

a b $a_i^{u_i}$ mod p $b_i^{u_i}$ mod p

i 1 i 1

$\mathbf{t}$ t_1 t_N t_i c_i^c g^{s_i} $c_{i\ 1}^{s_i}$ mod p

t_1 c^c g^{s_1} mod p

t_2 c^c g^{s_2} mod p

N

t_3 c^c g^{s_3} $h_i^{s_i}$ mod p

i 1

N N

$t_{4,1}$ $t_{4,2}$ b^c pk^{s_4} $b_i^{s_i}$ mod p a^c g^{s_4} $a_i^{s_i}$ mod p

i 1 i 1

c hash $\mathbf{e}$ $\mathbf{e}$ $\mathbf{c}$ $\mathbf{c}$ pk t_1 t_2 t_3 $t_{4,1}$ $t_{4,2}$ t_i t_N mod q

output: c c

Figure B.2: Shuffle Proof Verification

GetIndependentGenerators

known: p q g

inputs:

number of independent generators n

vote identifier id_{vote}

for i 1 N **do**

x 0

repeat

x x 1

h_i hash id_{vote} "const" i x mod p

h_i h_i^2 mod p

until h_i 0 1

output: h h_1 h_N h_i $_p$ 1

Figure B.3: Algorithm to Retrieve n Independent Generators for Vote id_{vote}.

GeneratePermutationCommitment

known: p q g

inputs:

permutation ψ j_1 j_N Ψ_N

Independent Generators **h** h_1 h_N h_i $_p$ 1

for i 1 N **do**

r_{j_i} r q

c_{j_i} $g^{r_{j_i}}$ h_i mod p

r r_1 r_N r_i $_q$

c c_1 c_N c_i $_p$

output: c r

Figure B.4: Algorithm to Generate Permutation Commitment for Permutation ψ.

GetChallenges

known: p q g

inputs:

number of independent generators n

encryptions **e** e_1 e_N e_i a_i b_i g^{r_i} pk^{r_i} m

shuffled encrpytions **e** e_1 e_N

permutation commitments **c** c_1 c_N c_i p

public key pk

H hash **e** **e** **c** pk

for i 1 N **do**

I hash i

u_i hash H I mod 2^τ τ 128

output: u u_1 u_N u_i q

Figure B.5: Algorithm to Generate n Challenge Values for a Set of Public Input values **e** **e** **c** pk.

GenerateCommitmentChain

known: p q g h

input: permuted public challenges **u** u_1 u_N u_i q

R_0 o U_0 1

for i 1 N **do**

r_i r q

R_i r_i u_i R_{i-1} mod q U_i u_i U_{i-1} mod q

c_i g^{R_i} h^{U_i} mod p

r r_1 r_N r_i q

c c_1 c_N c_i p

output: c r

Figure B.6: Algorithm to Generate Commitment Chain for the Permuted Public Challenges **u**.

C

Helverify Implementation Details

The following code excerpts and sequence diagrams are part of the Helverify implementation presented in Section 6.2.4 and based on [190].

```
/// <summary>
    /// Creates a proof that the specified ciphertext has been decrypted correctly.
    /// </summary>
    /// <param name="a">First component of an ElGamal ciphertext</param>
    /// <param name="b">Second component of an ElGamal ciphertext</param>
    /// <param name="publicKeyShare">Public key share</param>
    /// <param name="privateKeyShare">Private key share</param>
    /// <returns>Proof of correct decryption</returns>
    public static ProofOfDecryption Create(BigInteger a, BigInteger b,
        DHPublicKeyParameters publicKeyShare, DHPrivateKeyParameters privateKeyShare)
    {
        BigInteger p = publicKeyShare.Parameters.P;
        BigInteger q = p.Subtract(BigInteger.One)
            .Multiply(BigInteger.Two.ModInverse(p))
            .Mod(p);
        BigInteger g = publicKeyShare.Parameters.G;
        BigInteger pk = publicKeyShare.Y;
        BigInteger sk = privateKeyShare.X;

        BigInteger r = new BigInteger(privateKeyShare.X.BitLength, SecureRandom)
            .Mod(q);

        BigInteger u = a.ModPow(r, p).Mod(p);
        BigInteger v = g.ModPow(r, p).Mod(p);

        BigInteger c = HashHelper.GetHash(q, pk, a, b, u, v).Mod(q);

        BigInteger s = r.Add(c.Multiply(sk).Mod(q)).Mod(q);

        BigInteger d = a.ModPow(sk, p).Mod(p);

        return new ProofOfDecryption(d, u, v, s);
    }
```

Listing 4: Exemplary Proof Generation (Proof of Correct Decryption)

```
        /// <summary>
        /// Allows to verify that the specified ciphertext has been
        /// decrypted correctly.
        /// </summary>
        /// <param name="a">First component of an ElGamal ciphertext</param>
        /// <param name="b">Second component of an ElGamal ciphertext</param>
        /// <param name="publicKeyShare">Public key share</param>
        /// <returns>True if decryption has been performed correctly,
        /// false otherwise.</returns>
        public bool Verify(BigInteger a, BigInteger b,
            DHPublicKeyParameters publicKeyShare)
        {
            BigInteger p = publicKeyShare.Parameters.P;
            BigInteger q = p.Subtract(BigInteger.One)
                .Multiply(BigInteger.Two.ModInverse(p)).Mod(p);
            BigInteger g = publicKeyShare.Parameters.G;
            BigInteger pk = publicKeyShare.Y.Mod(p);

            BigInteger c = HashHelper.GetHash(q, pk, a, b, U, V).Mod(q);

            bool firstCondition = a.Mod(p).ModPow(S, p).Mod(p)
                .Equals(U.Multiply(D.ModPow(c, p).Mod(p)).Mod(p));

            bool secondCondition = g.Mod(p).ModPow(S, p).Mod(p)
                .Equals(V.Multiply(pk.ModPow(c, p).Mod(p)).Mod(p));

            return firstCondition && secondCondition;
        }
```

Listing 5: Exemplary Proof Verification (Proof of Correct Decryption)

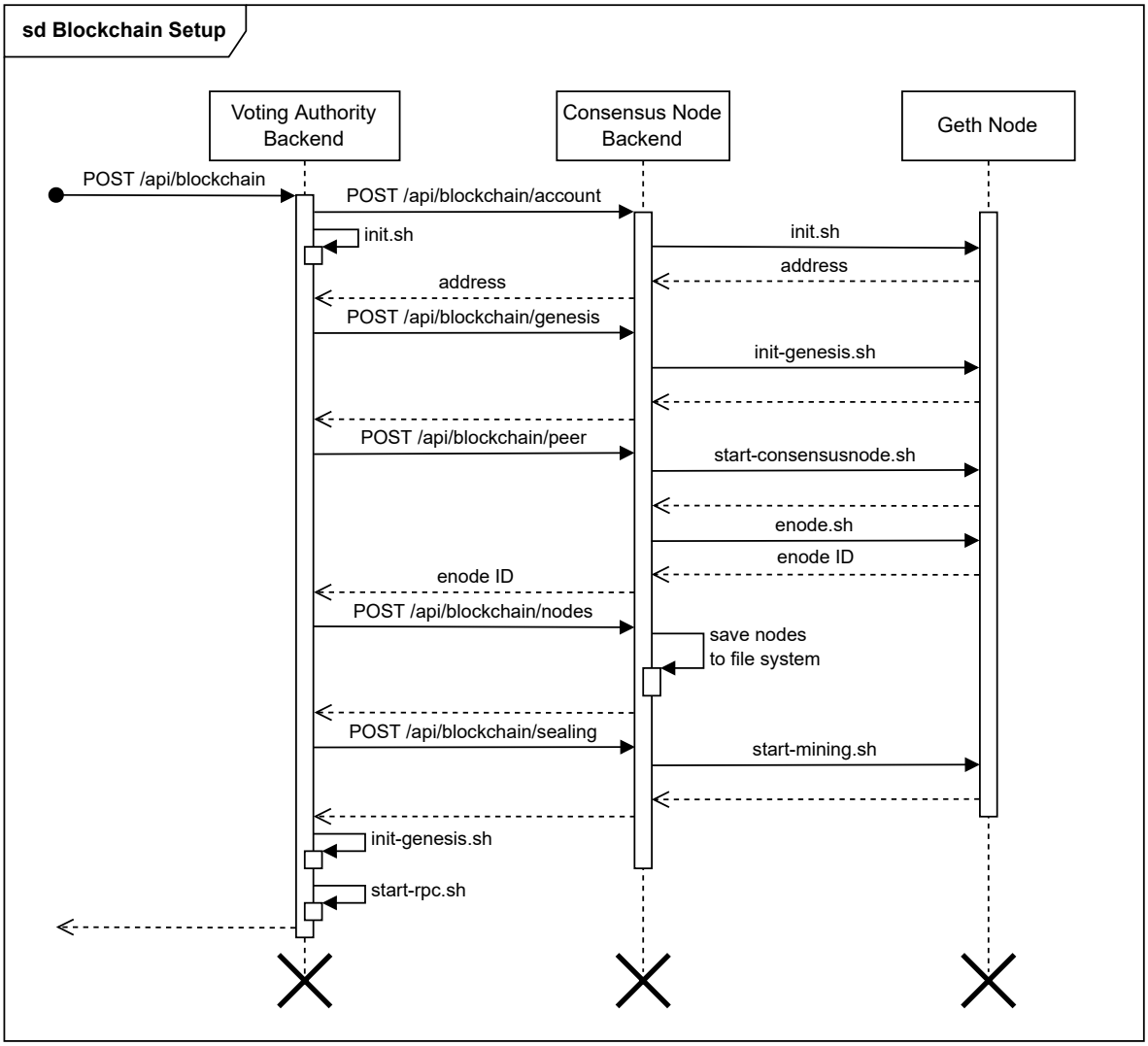


Figure C.1: Sequence Diagram showing the BC Setup Orchestrated by the VA Backend.

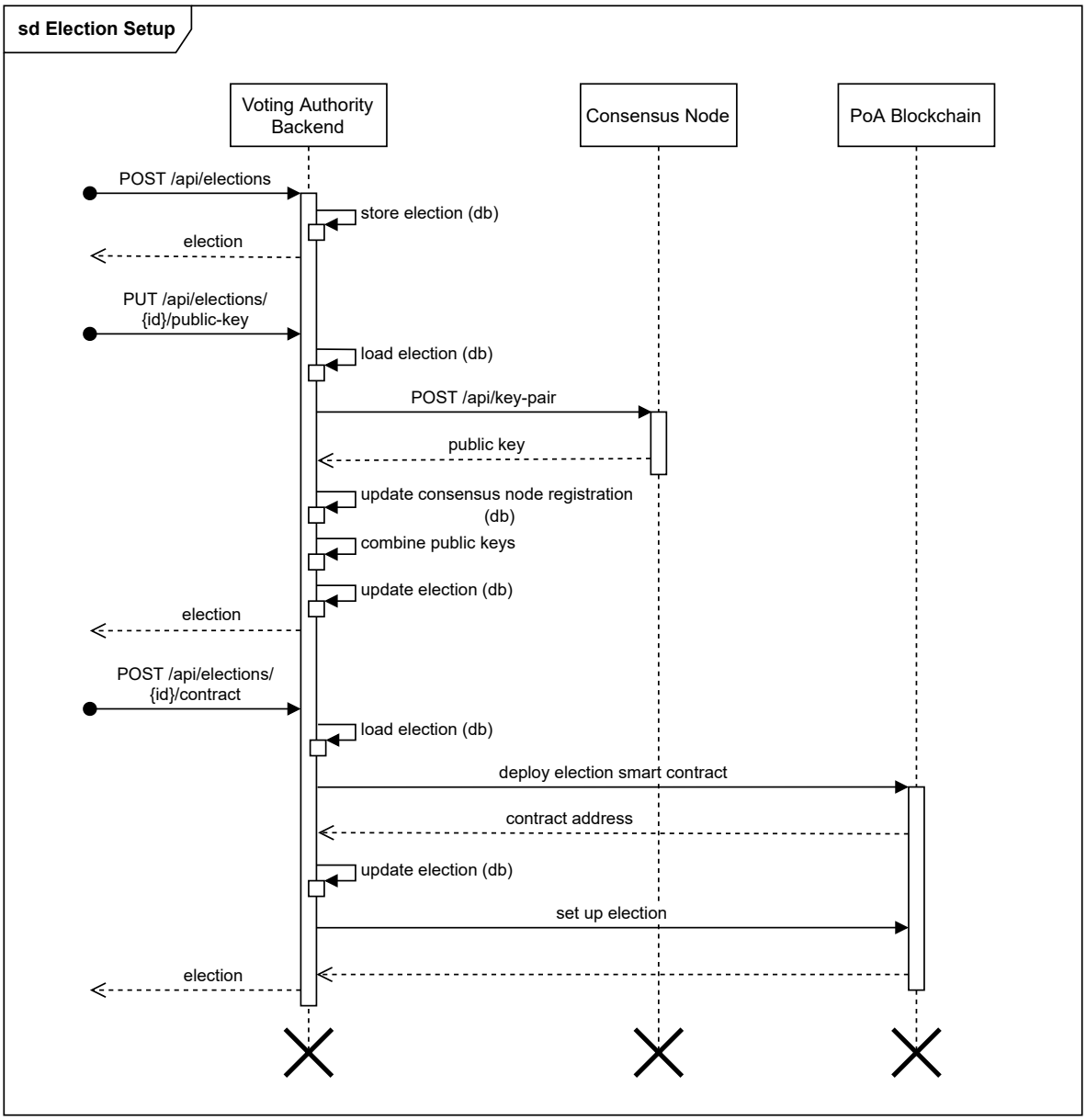


Figure C.2: Sequence Diagram Showing the Setup Process of an Election.

```
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="publicKey">Public key of the election</param>
        /// <param name="plainTextOption">Option in plaintext, list of zeros and a one
        /// (position of one = selected candidate / option)</param>
        public EncryptedOption(DHPublicKeyParameters publicKey, IList<int> plainTextOption)
        {
            Values = new List<EncryptedOptionValue>();

            foreach (int option in plainTextOption)
            {
                EncryptedOptionValue encryptedOptionValue
                    = new EncryptedOptionValue(option, publicKey);

                Values.Add(encryptedOptionValue);
            }

            HashHelper hashHelper = new HashHelper();

            Hash = hashHelper.Hash(Values.Select(v => v.Cipher).ToArray());

            ShortCode = Hash.Substring(0, 2);
        }
```

Listing 6: Short Code Calculation

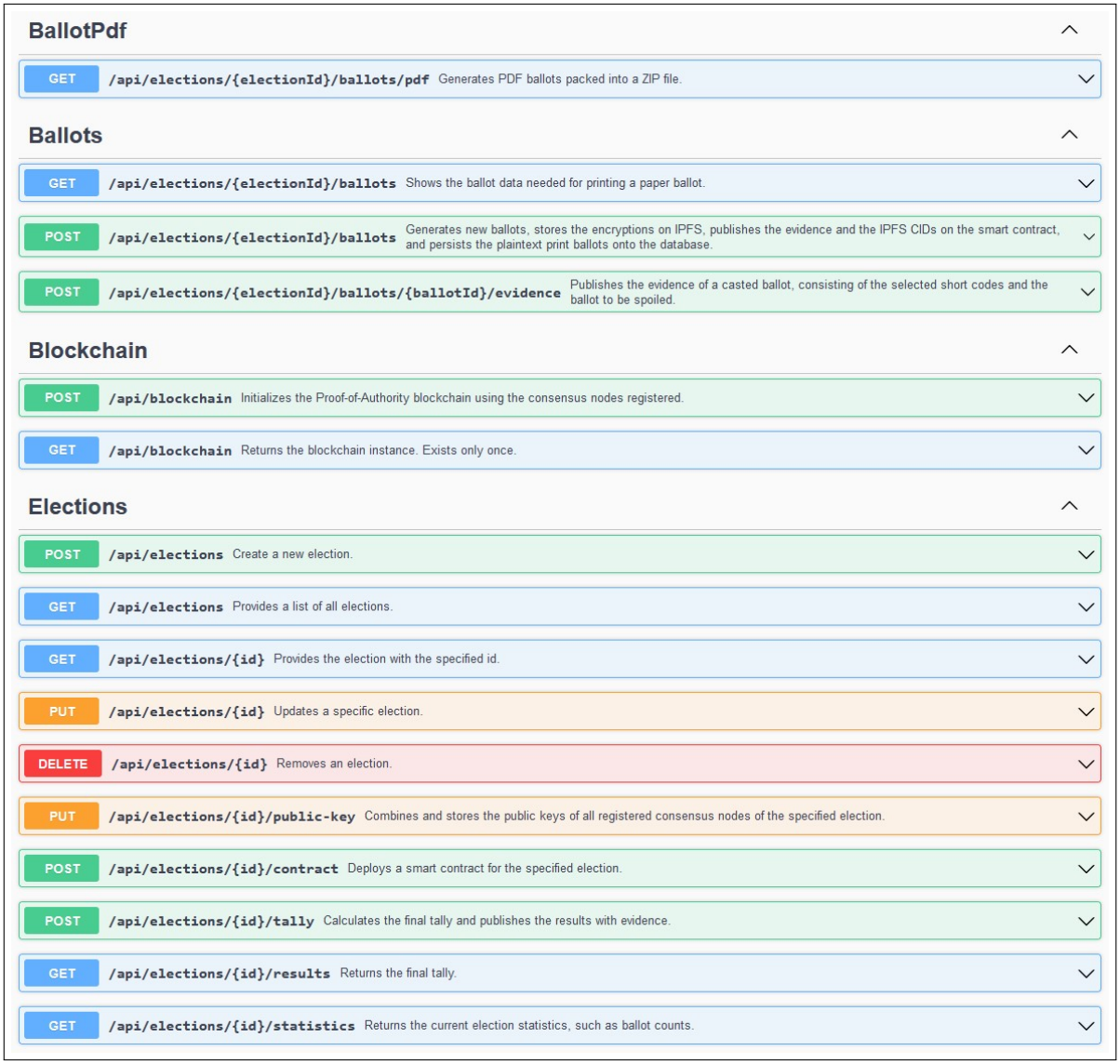


Figure C.3: Voting Authority Backend REST API

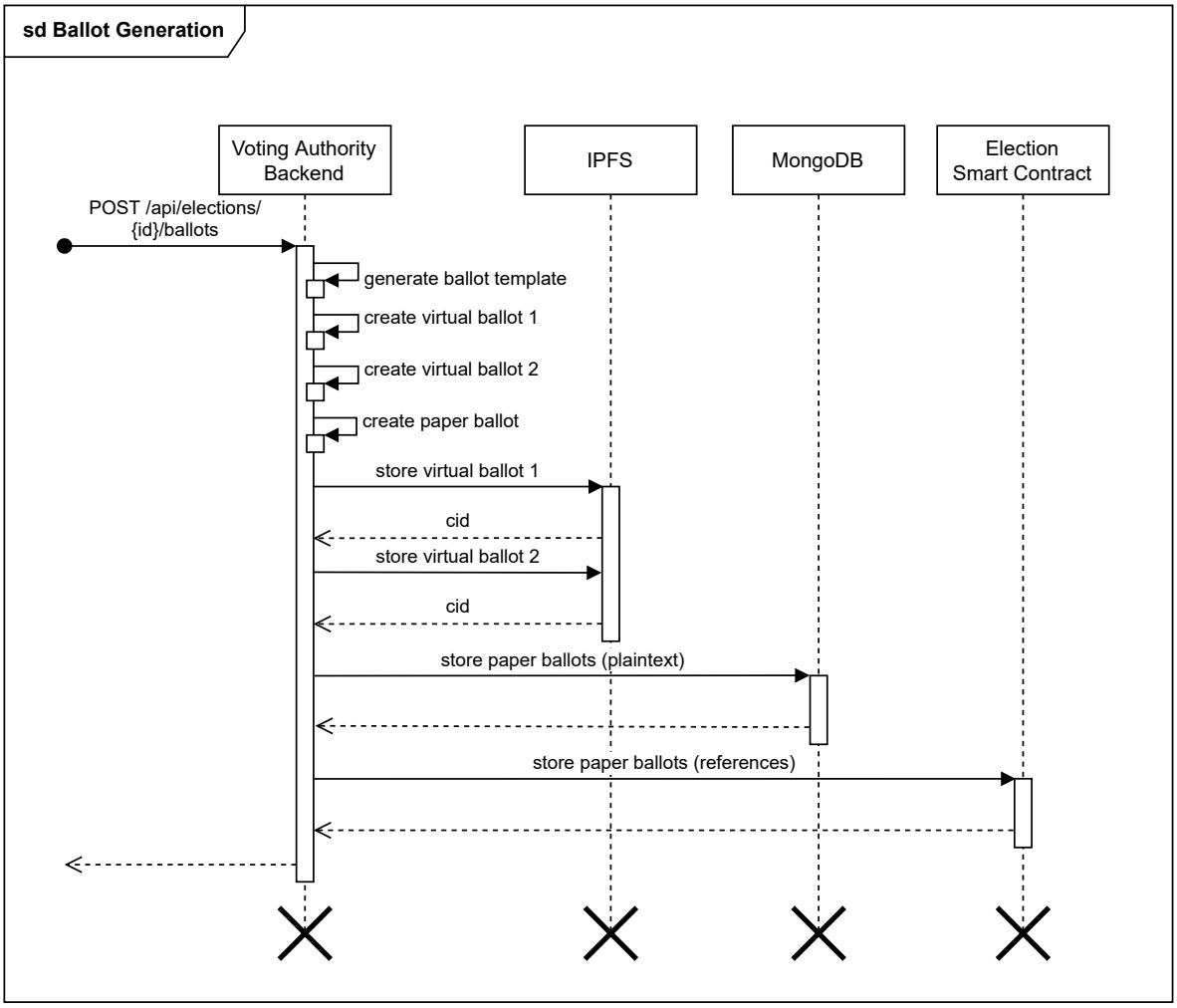


Figure C.4: Sequence Diagram Showing the Ballot Generation Process of an Election.

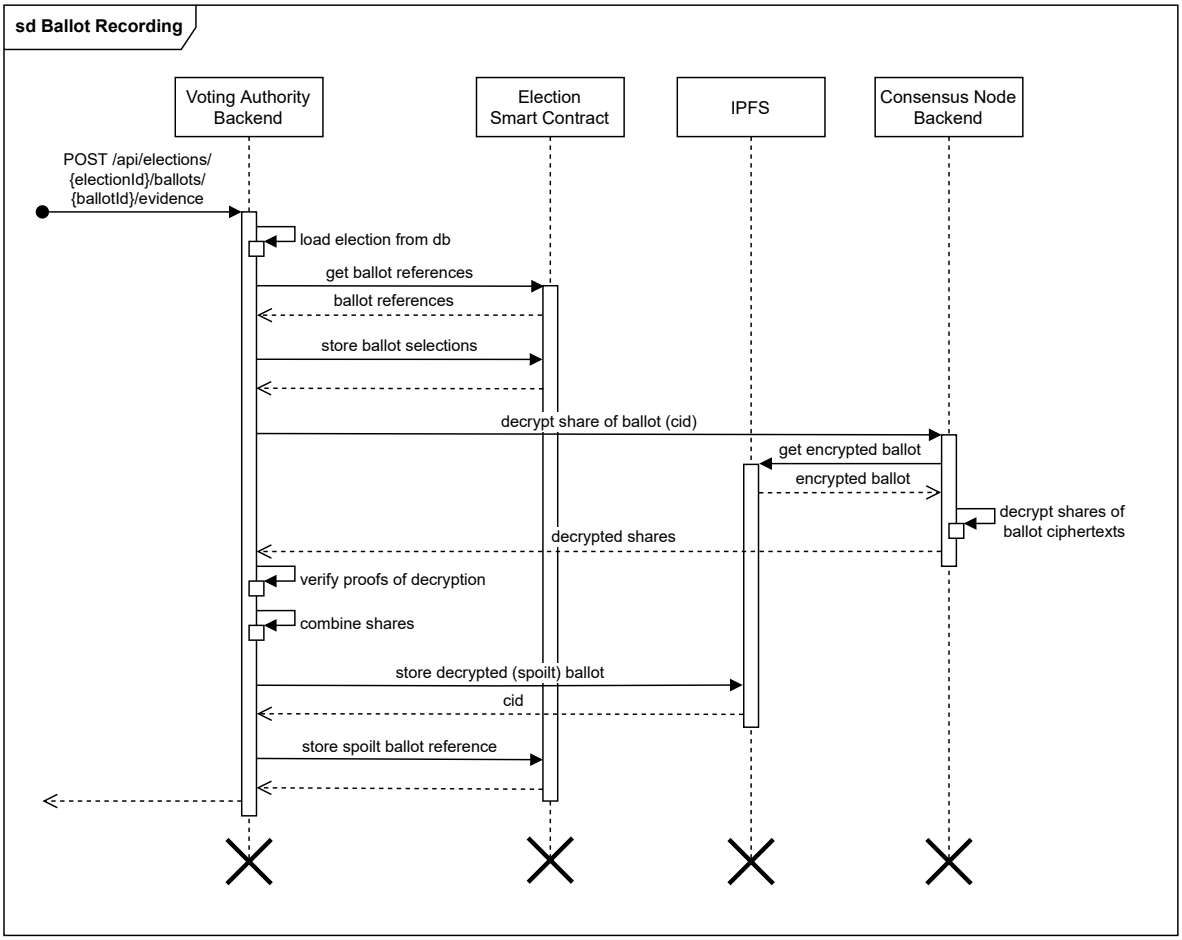


Figure C.5: Sequence Diagram Showing the Ballot Recording Process of an Election.

```
    /**
     * Verifies that the provided ciphertext indeed contains the value zero or one.
     * @param a C component of the ElGamal ciphertext
     * @param b D component of the ElGamal ciphertext
     * @param h Election public key
     * @param p ElGamal parameter p
     * @param g ElGamal parameter g
     */
    verify(a: bigInt.BigInteger, b: bigInt.BigInteger, h: bigInt.BigInteger,
        p: bigInt.BigInteger, g: bigInt.BigInteger) {
        const q: bigInt.BigInteger = p.subtract(bigInt.one)
            .multiply(bigInt(2).modInv(p)).mod(p);

        const c = HashHelper.getHash(q,
            [h, a, b, this.u0, this.v0, this.u1, this.v1]);

        const gInverse: bigInt.BigInteger = g.modInv(p).mod(p);

        const condition1: boolean = g.modPow(this.r0, p)
            .equals(this.u0.multiply(a.modPow(this.c0, p).mod(p)).mod(p));
        const condition2: boolean = g.modPow(this.r1, p)
            .equals(this.u1.multiply(a.modPow(this.c1, p).mod(p)).mod(p));
        const condition3: boolean = h.modPow(this.r0, p)
            .equals(this.v0.multiply(b.modPow(this.c0, p).mod(p)).mod(p));
        const condition4: boolean = h.modPow(this.r1, p)
            .equals(this.v1.multiply(b.multiply(gInverse)
            .mod(p).modPow(this.c1, p)).mod(p));
        const condition5: boolean = this.c0.add(this.c1).mod(q).equals(c);

        return condition1 && condition2 && condition3 && condition4 && condition5;
    }
```

Listing 7: Verification of Proof that a Ciphertext either contains the value 0 or 1.

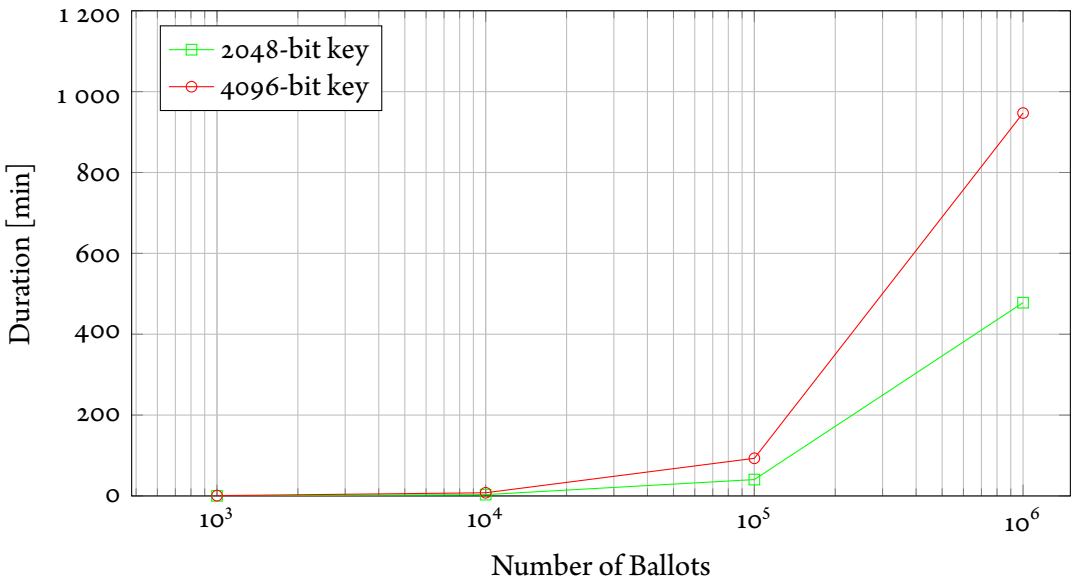


Figure C.6: Ballot Creation Duration for Different Key Lengths (Three Options)

D

Provotum 3.0 HE

The following algorithms and details are part of Provotum 3.0 HE presented in Section 7.1 and based on [148] and its subsequent extensions presented in Section 7.3, and 7.3.

D.1 Definitions

Definition D.1.1 (Proof of Re-encryption)**.** *A proof of re-encryption of an encrypted value e is a ZKP of equivalence of E o a e e a $_{q}$, where e is a re-encryption of e, i.e., e R e ξ [147].*

Definition D.1.2 (Designated-Verifier Proof of Re-encryption)**.** *A designated-verifier proof of re-encryption is a ZKP of re-encryption, composed of a logical disjunction of a Schnorr Proof, together with a re-encryption proof. The proof can only be verified by the voter, identified by the public-key pk_{voter} [147].*

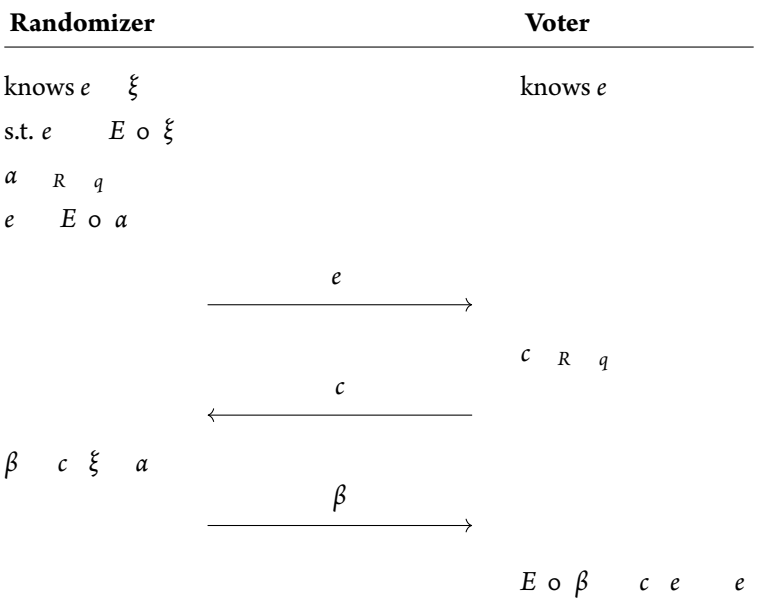


Figure D.1: Proof of $E\ v\ a \quad e \quad e \quad R\ e\ \xi$

D.2 Algorithms

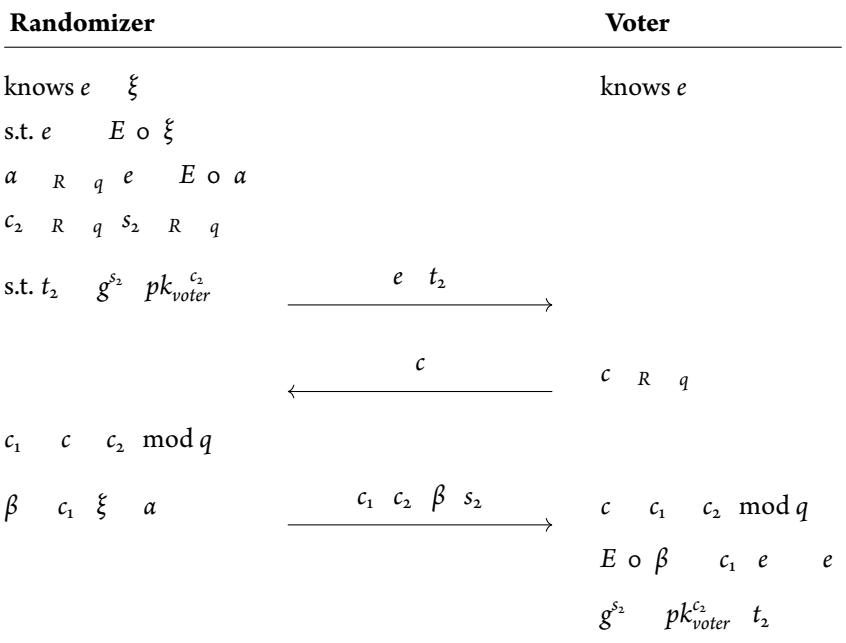


Figure D.2: Interactive Designated-Verifier Proof of Re-Encryption

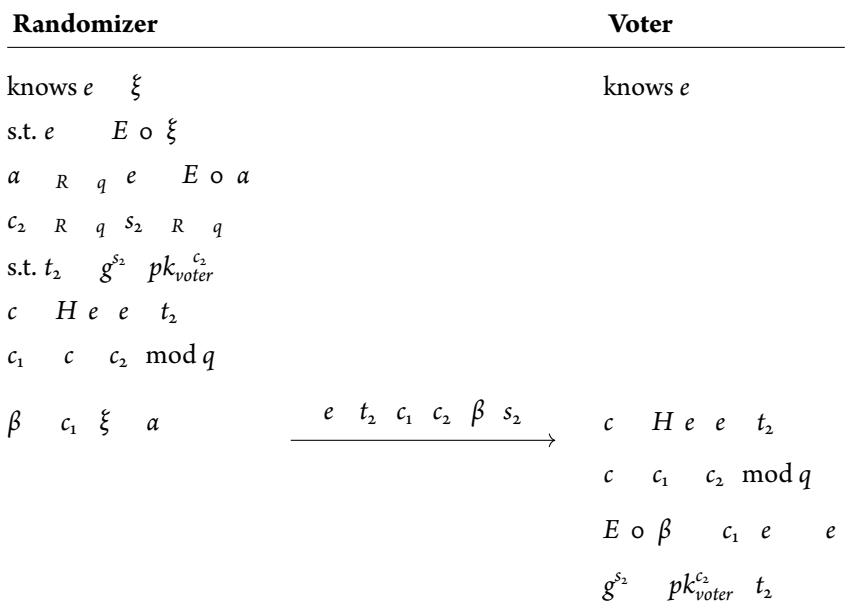


Figure D.3: Non-Interactive Designated-Verifier Proof of Re-Encryption

ID	TITLE	THREAT	MITIGATION
T2	Lack of RF	A dishonest voter can reconstruct her ballot creating a receipt because the randomization parameter used to encrypt the vote are available to her.	The Randomizer component adds RF to the scheme.
T3	Forced abstention	A coercer can stop a voter from voting by simply requiring the voter to submit her Ethers.	Native Tokens are not implemented present in the Substrate chain.

T4	Channel assumption	The protocol assumes the existence of an authenticated, untappable channel between sealers and VA. Without it, anyone could register their wallet as a sealer or run a MITM attack when fetching the BC's configuration to launch segmentation and 51% attacks.	Addressed in design.
T5	BC assumption	Reusing the same BC for multiple votes could allow a voter to participate in the DKG process. As shown in further findings (Table 5.6 in [148]), due to implementation issues, the adversary could block the vote and force a retry or manipulate the tally.	Addressed in design.
T10	Ballot weeding	The protocol does not specify any protection against ballot reuse. As shown by Cortier and Smyth [62], if an adversary were to copy ballots of a user, this can be exploited to determine who voted for which candidate. However, this attack is only applicable on very low-scale votes.	Added ballot weeding of any ballot, which is either an exact copy of the ciphertext or a DCP ZKP.
T11	Session hijacking	The token issued by the IdP can be stolen or bought. This impedes an eligible citizen from voting, while allowing an adversary to vote, bypassing authentication.	Removed use of the token.
T12	Token selling	The IdP token can be easily transferred to a vote buyer. Note that this is an issue with many remote voting systems, including postal vote.	Removed use of the token.
T15	IdP & AP collusion	Collusion of the IdP and AP would allow linking a ballot to an identity.	Removed AP.

T17	AP forges identities	A malicious AP could fund wallets of arbitrary users, allowing them to bypass authentication.	Removed AP.
T18	AP voting	A malicious AP could create and fund its own wallets, allowing it to bypass authentication.	Removed AP.

```
{
    "id":"Commitment",
    "BH":"3a965d1d44ef07c16904e9e09014100a27c3eee5e03a952f1065e5ea0dba1579",
    "Counter":0,
    "Total":1,
    "VotingQuestions":{

    "0x8d00badf7491ac322cd7969ab37e9b44632ba5fa7d47868fa113a128b6d40291":"Popular
    initiative 'For responsible businesses - protecting human rights and the
    environment'",

    "0xd386cf8378309431a880a92e0fc59339c331b802d7fa2409b90994a8689f16cc":"Popular
    initiative 'For a ban on financing war material manufacturers'"
    }
}
```

Listing 8: Commitment QR-code Content

```
{
    "id":"Challenge",
    "Key":"GeneralData",
    "Counter":0,
    "Total":5,
    "publicKey":{
        "h":"cb882ce4f7...",
        "parameters":{
            "p":"fffffffff...",
            "g":"02",
            "q":"7fffffffff..."
        }
    },
    "voterPublicKeyH":"6ae48794de..."
}
```

Listing 9: Challenge QR-code content 1

```
{
    "id":"Challenge",
    "Key":"0x8d00badf7491ac322cd7969ab37e9b44632ba5fa7d47868fa113a128b6d40291",
    "Counter":1,
    "Total":5,
    "answerBin":0,
    "Nonce":"3df6999519...",
    "reEncryptedBallot":{
        "c":"7094363b10...",
        "d":"a78b74de67..."
    }
}
```

Listing 10: Challenge QR-code Content 2

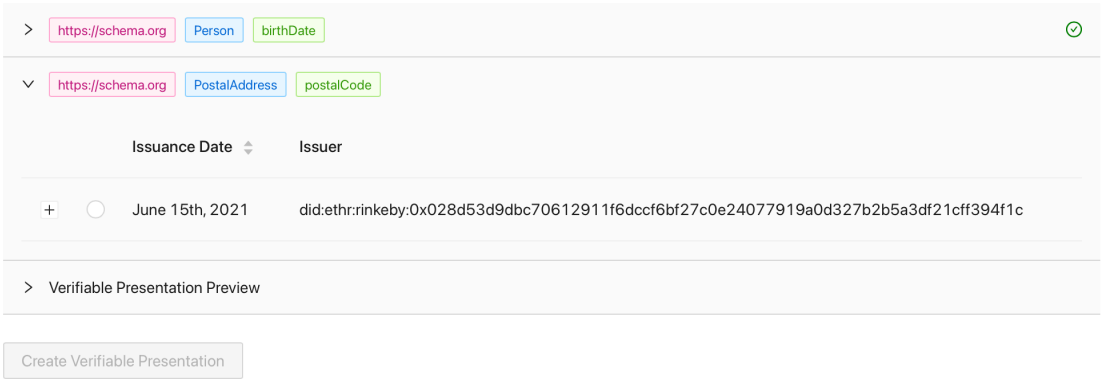


Figure D.9: UI for selectively disclosing VCs. After scanning the SDR, the voter is presented with a list of required credentials that must be disclosed. Each credential must be permitted to be disclosed.

```
{
    "id":"Challenge",
    "Key":"0x8d00badf7491ac322cd7969ab37e9b44632ba5fa7d47868fa113a128b6d40291",
    "Counter":2,
    "Total":5,
    "reEncryptionProof":{
        "ePrime":{
            "c":"770e7752c7...",
            "d":"2545a2c685...e"
        },
        "t2":"e1a1d01700...",
        "c1":"7f0127be2b...",
        "c2":"fed841d420...",
        "beta":"5bc5291701...",
        "s2":"9914178faa..."
    }
}
```

Listing 11: Challenge QR-Code Content 3

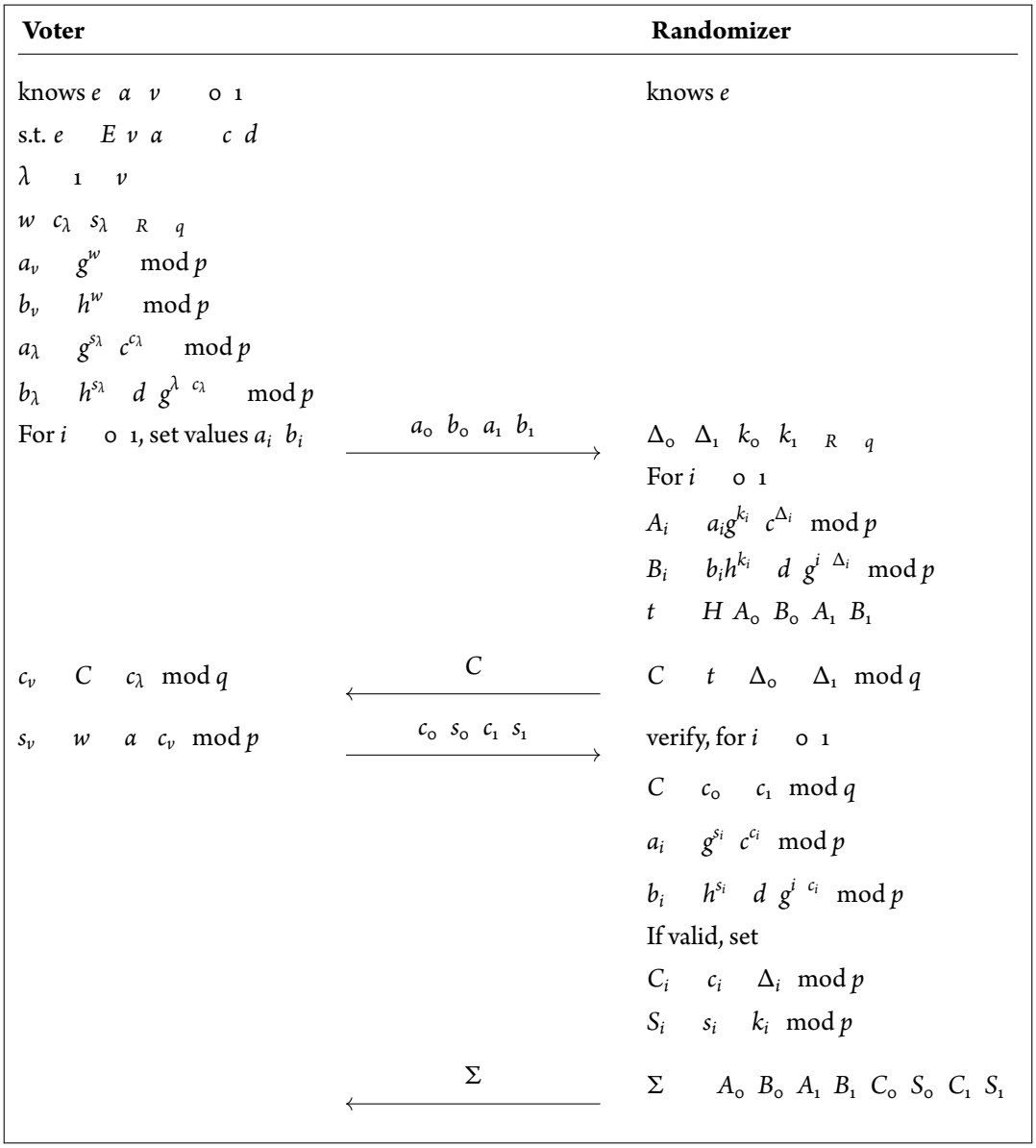


Figure D.4: Ballot proof (DCP) randomization

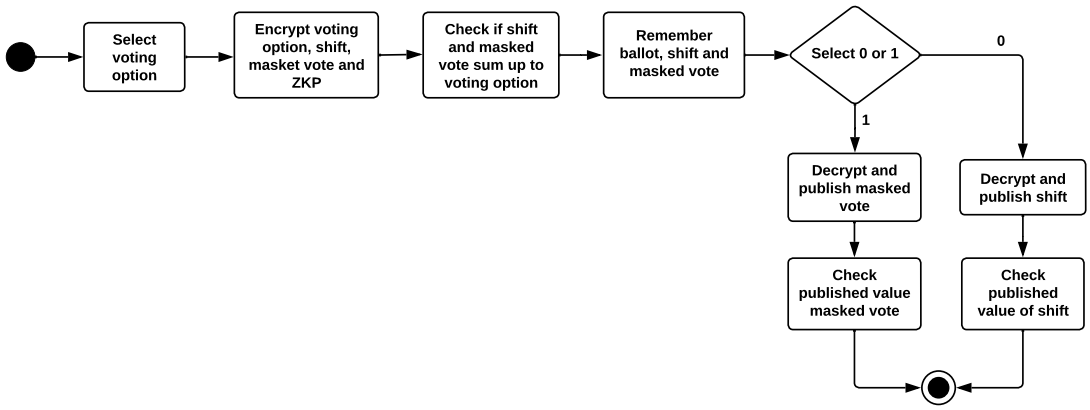


Figure D.5: The Partial-Audit Process (based on [58])

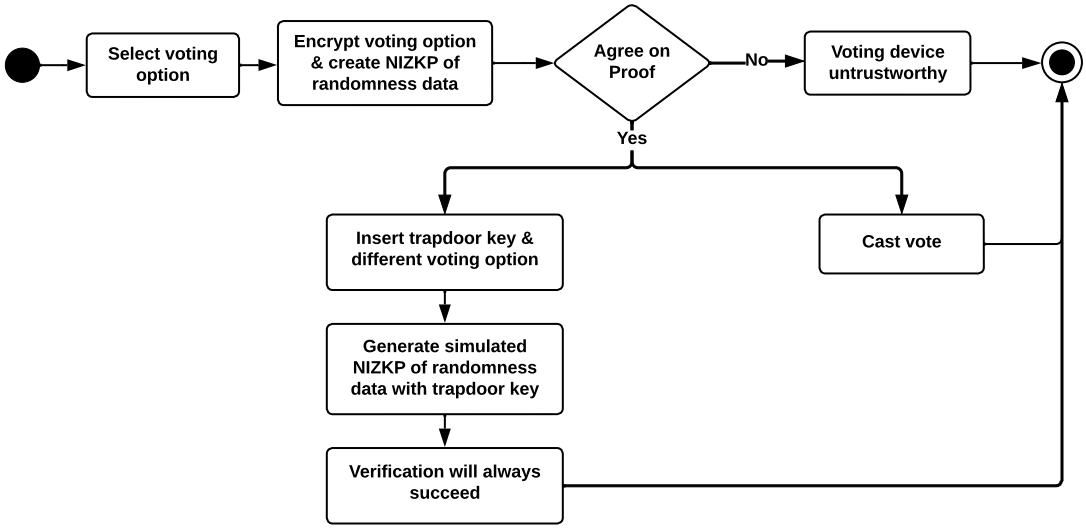


Figure D.6: The Challenge-and-Cast Process (based on [122, 123])

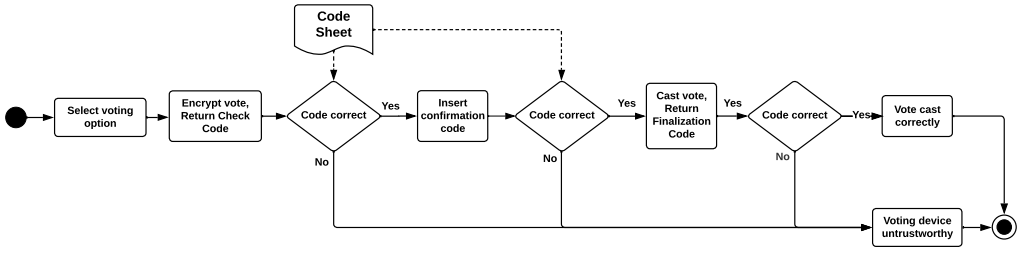


Figure D.7: The Code-Based Process (based on [195, 271])

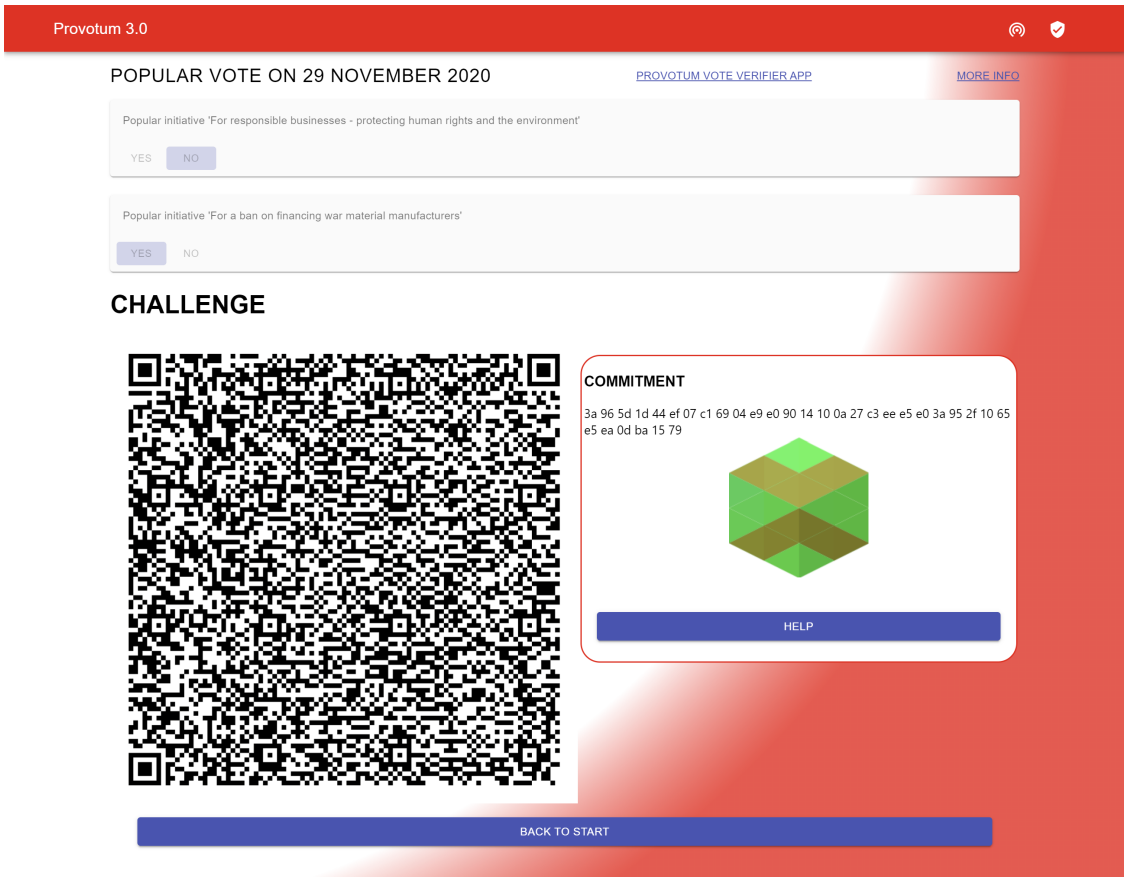


Figure D.8: The Challenge is Shown to the Voter in the Voting Front-End

Table D.2: Voting Scheme Threats Provotum 2.0

ID	Title	Threat
T1	Lack of CaI	CaI is achieved only under the assumption that the voting device is not compromised and acts correctly, which automatically negates the property.
T2	Lack of RF	A dishonest voter can easily reconstruct her own ballot to produce a receipt of her vote. It requires storing the randomization parameter used to encrypt the vote.
T3	Forced abstention	A coercer can stop a voter from voting by simply requiring the voter to submit her ether.
T4	Missing channel assumption	The protocol assumes the existence of an authenticated, untappable channel between sealers and VA. Without it, anyone could in practice register their wallet as a sealer or run a MITM attack when fetching BC's configs. This could be used to take over the network using segmentation attacks/51% attack.
T5	Missing BC assumption	Reusage of the same BC for multiple votes could allow a voter to participate in the distributed key generation process. As shown in further findings (table:impl-threats), due to implementation issues (not the protocol), the adversary could block the vote and force a retry or manipulate the tally.
T6	Attack of the clones	An attack to the consensus of PoA- or PoS-chains [93], which can be used to allow double spending. In this case, it could be used to prevent ballots from being included in the chain.
T7	Fault-tolerance	Due to the cooperative N N decryption scheme, if a single sealer loses the key, all ballots are lost and the tally cannot be computed.
T8	Sealer DoS	All sealers share their decrypted shares publicly and not at the same time. A malicious sealer could wait for all other sealers to submit, in order to compute the final tally on its own. Depending on the results, it could choose not to publish the last share, blocking the election.
T9	Decryption DoS	The time required to decrypt the final tally is directly proportional to the number of *yes* votes received. This could potentially be long-running and cause a denial-of-service on the network. For an election with a high turnout of voters, it would be better to compute the final tally offline and then publish the results on the BC.
T10	Lack of ballot weeding	The protocol does not specify any protection against ballot reuse. As shown by Cortier and Smyth [62], if an adversary were to copy the ballots of a user, this can be exploited to determine who voted for which candidate. However, this attack is only applicable on very low-scale votes.

Table D.3: Identity Management Threats of Provotum 2.0

ID	Title	Threat
T11	Session hijacking	The token issued by the IdP can be stolen or bought. This impedes an eligible citizen from voting, while allowing an adversary to vote, bypassing authentication.
T12	Token selling	The IdP token can be easily transferred to a vote buyer. Note that this is an issue with many remote voting systems, including postal vote.
T13	Voter impersonation	If a voter registers for electronic voting, but does not vote, the IdP can vote on the voter's behalf, by using her assigned token.
T14	Token re-use	The tokens provided from the IdP may not be reused in other elections. An adversary observing multiple elections could track usage of the same tokens to link votes to the same pseudonymous identity, which provides useful information for de-anonymisation through statistical analysis.
T15	IdP & AP collusion	Collusion of the IdP and AP would allow linking a ballot to an identity.
T16	IdP forges identities	A malicious IdP could forge identities by minting more one-time tokens.
T17	AP forges identities	A malicious AP could fund wallets of arbitrary users, allowing them to bypass authentication.
T18	AP voting	A malicious AP could create and fund its own wallets, allowing it to bypass authentication.

Table D.4: Implementation Threats of Provotum 2.0

ID	Title	Threat
T19	Missing authorization	The SC does not verify whether the public-key shares are coming from sealers or not. Anyone who sends a transaction with a valid public-key and proof, will take part in the cooperative scheme. The implicit assumption here is that no non-sealer user would have the funds necessary to submit transactions to the BC before the Voting phase. A colluding sealer or AP could transfer funds to a wallet address to allow someone else to send transactions. This could be used during the tallying phase to block the final tally decryption.
T20	Tampering	Due to an implementation flaw, it is possible for the last sealer to tamper with the final results. This could be detected by auditing all SC function calls, which is a plausible assumption to make, depending on the vote (low/high stakes). The flaw is that the SC does not compute the homomorphic sum but simply uses the one submitted by the sealers. In the case of honest sealers, the sum should always be the same. The SC also stores the last submitted homomorphic sum to use for the final decryption (as a way to not have to compute the homomorphic sum in the SC). A malicious sealer, given all decrypted shares, could attempt to find a different encrypted sum, such that when decrypted and combined, returns a final tally to the favor of some candidate.

```
function CreateEncryptedBallot(votingQuestions: Array<any>, publicKey:
    ElGamalPublicKey, voterPublicKeyH: BN) {
  var allVerified: any = null
  var verifies: Boolean = false

  const encryptedBallots: Array<any> = []

  Object.entries(votingQuestions).forEach(([key, value]) => {
    if (value.answerBin === undefined) {
      return
    }
    const encryptedVote: Array<any> = []

    const encryptedBallot = encrypt(value.answerBin, publicKey, value.nonce);

    var verifies = verifyReEncryptionProof(
      value.reEncryptionProof,
      value.reEncryptedBallot,
      encryptedBallot,
      publicKey,
      voterPublicKeyH,
    );

    allVerified = allVerified !== null ? (verifies && allVerified ? (true) :
      (false)) : (verifies)

    const cipherToSubstrate = {
      c: bnToHex(value.reEncryptedBallot.c),
      d: bnToHex(value.reEncryptedBallot.d),
    };

    encryptedVote.push(key);
    encryptedVote.push(cipherToSubstrate);
    encryptedVote.push(encryptedBallot);
    encryptedBallots.push(encryptedVote);
  })

  verifies = allVerified !== null ? (allVerified) : (false);
  return [encryptedBallots, verifies]
}
```

Listing 12: Encryption Process for the Verifier

List of Figures

1.1 Global Map of REV Adoption per 2016, from [186] 4
1.2 Structure of PhD Thesis 10

2.1 Overview of Major Cryptographic Primitives (introduced in Chapter 2), in Their Relation to REV Approaches (introduced in Chapter 3) 12
2.2 Basic Symmetric Encryption and Decryption Algorithms 12
2.3 Basic Asymmetric Encryption and Decryption Algorithms 13
2.4 Attack Models and Attacker's Capabilities 14
2.5 Secret Sharing 22
2.6 Types of Mixnets 27
2.7 Pseudo-Code Permutation Algorithm 28
2.8 Pseudo-Code Shuffle Algorithm 28
2.9 Three-Move Σ-Protocol 29
2.10 Schnorr Proof for Key Generation 32
2.11 Decryption Proof 34
2.12 Designated Verifier Re-Encryption Proof 37

3.1 Overview of Protocol Requirements 42
3.2 Timeline of Privacy Notions 43
3.3 Overview of System Requirements 47
3.4 Centralized REV System's Architecture 49
3.5 A Fully Decentralized and Generic High-level REV System's Architecture 51
3.6 Overview of REV System Development Phases 53
3.7 Deployment Models for Architectures [177] 57
3.8 Basic Remote E-Voting Architecture based on a Simple Encrypt and Sign 58
3.9 Two Agencies Model with Anonymous Tokens (*e.g.*, Blind Signatures) 59
3.10 Homomorphic Encryption-based REV System (*e.g.*, ElGamal) 60
3.11 Mix Network-based REV System 61

4.1 Keyword Analysis 63
4.2 CGS97 Domain Protocol 64
4.3 JCJ05 Protocol and Dependent Implementations 69
4.4 HRZ08 Domain Protocol and its Dependent Implementation 70

4.5 Example of in RemoteVote [66] . . . 78
4.6 Example of in SAFE Vote [66] . . . 79

5.1 Swiss Legal Framework . . . 83
5.2 Paper Voting Process Phases . . . 85
5.3 Paper Voting Process Flow (PVPF) . . . 86
5.4 Abstract Representation of the Necessary Paper Artifacts . . . 87
5.5 Threat Event Interdependencies . . . 95

6.1 Proverum System Design for Hybrid Public Verifiability . . . 101
6.2 Proverum Private Environment Architecture . . . 102
6.3 Overview of the Proverum Prototype . . . 104
6.4 Prototype Network Architecture . . . 105
6.5 Class Diagram . . . 106
6.6 *UC1:* Citizen Relocation . . . 107
6.7 *UC2:* Generation and Validation of Electoral Registers (ER) . . . 108
6.8 *UC3:* Secured Accumulation and Publication of Preliminary Results . . . 108
6.9 Use Cases of Helverify . . . 113
6.10 Helverify Protocol Phases . . . 115
6.11 Simplified Process Flow of Running an Election in Helverify . . . 115
6.12 Helverify Protocol Setup Phase . . . 116
6.13 Helverify Protocol Ballot Creation Phase . . . 117
6.14 Helverify Protocol Delivery Phase . . . 117
6.15 Helverify Protocol Casting Phase . . . 118
6.16 Helverify Protocol Storage Phase . . . 118
6.17 Helverify Protocol Tallying, Verification, and Destruction Phases . . . 119
6.18 Architecture of Helverify . . . 120
6.19 Consensus Node Backend REST API . . . 122
6.20 Helverifyclass diagram . . . 124
6.21 Evidence Publication . . . 126
6.22 Voting Authority UI . . . 127
6.23 Printed Ballot . . . 128
6.24 Ballot Authenticity Verification . . . 130
6.25 Choice and Spoilt Ballot Verification . . . 131
6.26 Verification Summary . . . 132
6.27 Number of NIZKPs required for number of options/candidates . . . 134
6.28 Ballot creation duration for different number of options (2048-bit key length) . . . 134
6.29 Tallying duration for different key lengths (3 options) . . . 135
6.30 Tallying duration for different number of options (2048-bit key length) . . . 136
6.31 Helverify's trust boundaries (notation inspired by [275]) . . . 138

7.1 Overview of Decentralized REV systems introduced in Chapter 7 . . . 145
7.2 Provotum 3.0 HE architecture . . . 146

7.3 Provotum 3.0 scheme's phases . . . 147
7.4 Registration of Voters with blind signatures . . . 148
7.5 Voter, Randomizer and PBB Interaction . . . 150
7.6 Runtime for verifying RSA signatures . . . 158
7.7 Runtime for verifying DCP proofs . . . 159
7.8 Runtime for the homomorphic sum of ballots . . . 160
7.9 Runtime for decrypting the tally of ballots . . . 160
7.10 Challenge-or-Cast Process . . . 164
7.11 Theoretical QR-Code content for commitment and challenge . . . 170
7.12 Provotum Trust Boundaries . . . 173
7.13 Challenge-or-Cast Process in detail . . . 175
7.14 Provotum QR-Code content for commitment and challenge . . . 176
7.15 Commitment displayed in voting front end . . . 178
7.16 Verifier Intro Page . . . 180
7.17 Verifier Intro Page with Help . . . 180
7.18 Commitment Scanner . . . 180
7.19 Successful Commitment Scan . . . 180
7.20 Scanning of the Challenge . . . 180
7.21 Scanning Error . . . 180
7.22 Confirmation of Selection . . . 181
7.23 Successful Verification . . . 181
7.24 Unsuccessful Verification . . . 181
7.25 DID Model of Issuer, Holder and Verifier . . . 192
7.26 Comparison of Account and SSI-based model . . . 193
7.27 DIDs as Self-Certifying Identifiers . . . 195
7.28 DID Documents Key Rotation . . . 196
7.29 Relationship between Identifier, Metadata and the Subject . . . 200
7.30 Original Identity Context Ends Whenever a Proxy Identity is Created. . . . 203
7.31 Provotum 3.0 Extended with A Decentralized IdM System . . . 209
7.32 Overview of Provotum's SSI repositories . . . 214
7.33 Credential Issuance UI . . . 217
7.34 Claim Auth UI . . . 218
7.35 Sequence diagram for IdP authentication process . . . 220
7.36 Governance Framework . . . 227
7.37 Credential registry for public VCs . . . 227
7.38 ProvotuMn's Voting Cryptographic Building Blocks . . . 228
7.39 ProvotuMn Stakeholders . . . 230
7.40 ProvotuMn's Voting Scheme . . . 231
7.41 Pre-Voting Phase, Registration Step . . . 232
7.42 Pre-Voting Phase, Key-Pair and Proof Gen . . . 233
7.43 Voting Phase . . . 234
7.44 Post-Voting Phase, Tallying . . . 235

7.45 Post-Voting Phase, Results . . . 235
7.46 Dependencies of Æternum's Assumptions A1 to A4 . . . 247
7.47 The Four Protocols of the UP Scheme [203, 204] . . . 248
7.48 Æternum System Architecture. . . . 253

B.1 Shuffle Proof Generation . . . 299
B.2 Shuffle Proof Verification . . . 300
B.3 Algorithm to Retrieve n Independent Generators for Vote id_{vote}. . . . 301
B.4 Algorithm to Generate Permutation Commitment for Permutation ψ. . . . 301
B.5 Algorithm to Generate n Challenge Values for a Set of Public Input values $\mathbf{e}$ $\mathbf{e}$ $\mathbf{c}$ pk. 302
B.6 Algorithm to Generate Commitment Chain for the Permuted Public Challenges $\mathbf{u}$. . 302

C.1 Sequence Diagram showing the BC Setup Orchestrated by the VA Backend. . . . 306
C.2 Sequence Diagram Showing the Setup Process of an Election. . . . 307
C.3 Voting Authority Backend REST API . . . 309
C.4 Sequence Diagram Showing the Ballot Generation Process of an Election. . . . 310
C.5 Sequence Diagram Showing the Ballot Recording Process of an Election. . . . 311
C.6 Ballot Creation Duration for Different Key Lengths (Three Options) . . . 313

D.1 Proof of $E\ v\ a\quad e\quad e\quad R\ e\ \xi$ . . . 315
D.2 Interactive Designated-Verifier Proof of Re-Encryption . . . 315
D.3 Non-Interactive Designated-Verifier Proof of Re-Encryption . . . 316
D.9 UI for selectively disclosing VCs . . . 319
D.4 Ballot proof (DCP) randomization . . . 321
D.5 Partial-Audit Process . . . 322
D.6 Challenge-and-Cast Process . . . 322
D.7 Code-based Process . . . 322
D.8 Challenge Displayed in Voting Front-End . . . 323

List of Listings

1 Creation of the commitment 177
2 Default DID Document Without Transaction History from Ethereum DID-Resolver 215
3 Example authentication configuration 218
4 Exemplary Proof Generation (Proof of Correct Decryption) 304
5 Exemplary Proof Verification (Proof of Correct Decryption) 305
6 Short Code Calculation 308
7 Verification of Proof that a Ciphertext either contains the value 0 or 1. 312
8 Commitment QR-code Content 318
9 Challenge QR-code content 1 319
10 Challenge QR-code Content 2 319
11 Challenge QR-Code Content 3 320
12 Encryption Process for the Verifier 326

List of Tables

4.1 Comparison of Related Work on REV Systems . . . 65
4.2 Comparison of PBBs in Terms of BC / DL Type . . . 74
4.3 Comparison of RPV Systems . . . 75

5.1 Official German Terminology with Corresponding English Translations . . . 84
5.2 Threat Events on the Swiss Postal Voting Process Flow (PVPF) . . . 92

6.1 Comparison of Proverum's Public Environments . . . 103
6.2 Threat Events and Mitigation Techniques of the Swiss RPV [180] . . . 110
6.3 Duration measurements in minutes for creating ballots . . . 133
6.4 Duration measurements in minutes for tallying ballots . . . 135
6.5 Projected durations: Vote on Covid 19 law (13.06.2021) [37] . . . 142
6.6 Projected durations: Vote on changes to Covid 19 law (28.11.2021) [36] . . . 143

7.1 Cryptographic primitives applied in Provotum 3.0 HE . . . 148
7.2 Number of modExp operations by entity, in order to cast a single *yes* or *no* vote. . . . 157
7.3 Verification Mechanisms Comparison . . . 167
7.4 Security Threats Analysis . . . 183
7.5 Mitigation strategies . . . 184
7.6 Types of DIDs . . . 197
7.7 Reserved Tokens in the JSON-LD Namespace that are Relevant for this Solution . . . 198
7.8 Threat Model of Provotum 3.0 in Regards to IdM . . . 207
7.9 Threat Model Evaluation after applying the new Design . . . 221
7.10 ProvotuMn's Assumptions . . . 231
7.11 Comparison of Privacy Properties . . . 236
7.12 Visualization of Verifiability Properties comparing Provotum 2.0 and ProvotuMn 3.0 237
7.13 Visualization of the practical properties across all three Provotum projects. . . . 238
7.14 Comparison of Cryptographic Operations . . . 241
7.15 Performance Difference of Message Encoding Strategies . . . 242
7.16 Ballot Shuffling Performance . . . 243
7.17 Mixnet Performance [s] for p of Size 2 048 *bits*. . . . 244
7.18 Decryption Proof Performance . . . 244
7.19 Mixing Times [s] in the Distributed Deployment . . . 245

7.20 Runtime [s] for randomizing the ballots 246
7.21 Æternum's Assumptions 248
7.22 Average Proof Generation Time [ms] 255
7.23 Average Proof Verification Time [ms] 255
7.24 Cryptoperiods' Security Level Recommendations 257

A.1 List of Publications per Chapter 294

D.2 Voting Scheme Threats Provotum 2.0 324
D.3 Identity Management Threats of Provotum 2.0 325
D.4 Implementation Threats of Provotum 2.0 325

Curriculum Vitae

Personal Details

Name	Christian Killer
Date of Birth	June 18, 1991
Place of Birth	Baden, Aargau, Switzerland

Education

Feb 2019 – Jul 2023	Doctoral program at the University of Zurich, UZH
	Department of Informatics, IfI, Communication Systems Group, CSG
	University of Zurich, UZH
Feb 2016 – Jan 2018	Master of Science in Informatics (M. Sc.)
	Major: Information Systems, Minor: Informatics
	University of Zurich, UZH
Aug 2012 – Jan 2016	Bachelor of Science in Informatics (B.Sc.)
	Major: Information Systems
	University of Zurich, UZH

Professional Experience

June 2023 – present	Head of Research, *Acurast Association*
Mar 2023 – present	Co-Founder and Security Expert, *prtcl GmbH*
Feb 2019 – Mar 2023	Junior Researcher, *Communication Systems Group (CSG), University of Zurich, UZH*
Feb 2017 – Oct 2022	Technical Freelancer, *Killer Design EF*
Aug 2015 – Dec 2016	Web Specialist in Multichannel Strategy, *UBS Switzerland AG*