2021

Open Voting Client Architecture and Op-Ed Voting: A Novel Framework for Solving Requirement Conflicts in Secret Ballot Elections

Aaron M. Wilson
University of North Florida, n01417717@unf.edu

OPEN VOTING CLIENT ARCHITECTURE AND OP-ED VOTING: A NOVEL FRAMEWORK FOR SOLVING REQUIREMENT CONFLICTS IN SECRET BALLOT ELECTIONS

by

Aaron Wilson

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

December, 2021

The thesis "Open Voting Client Architecture and Op-Ed Voting: A Novel Framework for Solving Requirement Conflicts in Secret Ballot Elections" submitted by Aaron Wilson in partial fulfillment of the requirements for the degree of Master of Science in Computing and Information Sciences has been

Approved by the thesis committee:	Date:
Dr. Swapnoneel Roy Thesis Advisor and Committee Chairperson	
Dr. Asai Asaithambi Committee Member	
Dr. Sandeep Reddivari Committee Member	

ACKNOWLEGEMENTS

There are many people around me without whom I would not have been able to reach this milestone in my life. First and foremost, I would like to give special thanks to my wife and my kids who were very supportive and patient with me. I also would like to give special thanks to my thesis advisor Dr. Swapnoneel Roy and to my advising committee: Dr. Asai Asaithambi and Dr. Sandeep R. Reddivari, who supported me in my academic journey. Their advice, encouragement, patience, and great feedback helped my advancement at the personal level as well as the professional level, which in turn helped improve the research work significantly. Also, I would like to thank Cyber Florida as this research work was supported in part by their grant (#220384). Lastly, I would like to thank all professors at the University of North Florida who taught me during my master's program, as I have learned so much from each of them.

CONTENTS

List of Figures ix

List of Tables x

Abstract xi

Chapter 1 INTRODUCTION 1

Chapter 2 Voting System Requirements and Conflict Model 3

2.1 Our Approach. 4

2.2 Electronic Voting Goals 6

2.2.1 Secret Ballot Goal 6

2.2.2 One Person One Vote Goal 7

2.2.3 Universal Access Goal 7

2.2.4 Transparent and Auditable Goal 9

2.3 Electronic Voting Goal Conflict Model 10

2.3.1 Voter Authentication Interferes with Voter Anonymity . . . 11

2.3.2 Voter Authentication Interferes with Voter Usability. 13

2.3.3 Voter Anonymity Diverges with Voter Accessibility 13

2.3.4 Coercion Resistance Interferes with Voter Usability 15

2.3.5 Coercion Resistance Diverges with Cast as Intended 16

2.3.6 Coercion Resistance Diverges with Recorded as Cast 17

2.3.7 Provisional Voting Diverges with Voter Anonymity 18

2.3.8 Cast as Intended Interferes with Voter Usability 19

2.3.9 Cast as Intended Interferes with Voter Accessibility 20

2.3.10 Recorded as Cast Diverges with Voter Anonymity 21

2.3.11 Tallied as Recorded Diverges with Voter Anonymity 21

Chapter 3 Voting System Reference Architecture 23

3.1 Definition 23

3.2 Components 23

3.2.1 ElectionDesigner 23

3.2.2 BallotDesigner 23

3.2.3 BallotProvider 24

3.2.4 VotingClient 24

3.2.5 BallotProcessor 25

3.2.6 BallotTabulator 25

3.2.7 ElectionReporter 25

3.2.8 Registrar 25

Chapter 4 Open Voting Client Architecture 26

4.1 Definitions 26

4.2 Description 27

4.2.1 iRegistrar 28

4.2.2 iBallotProvider 28

4.2.3 iBallotProcessor 29

4.3 Architectural Comparisons 29

4.3.1 Paper Ballot Voting Systems 29

4.3.2 Electronic Voting Systems. 30

4.4 Benefits 30

4.4.1 Improved Usability 31

4.4.2 Increased Comfort/Familiarity 31

4.4.3 Flexible Accessibility 31

4.4.4 Reduced Costs 31

4.5 Security Risks 32

Chapter 5 Op-Ed Voting Implementation 33
5.1 End-to-end Verifiable Protocol Selection 34
5.2 About ElectionGuard 35
5.3 ElectionGuard Crytographic Techniques and Key Concepts 36
5.3.1 Exponential ElGamal 36
5.3.2 Threshold Encryption 37
5.3.3 Schnorr Proof. 38
5.3.4 Chaum-Pedersen Proof 38
5.3.5 Cramer-Damgård-Schoenmakers Technique 39
5.3.6 Fiat-Shamir Heuristic 40
5.3.7 Tracking Codes 40
5.3.8 Challenge Ballots. 40
5.3.9 Election Record 41
5.4 Op-Ed Components and Voting Workflow 41
5.4.1 Op-Ed Voting Clients 41
5.4.2 Voter Authentication 42
5.4.3 Ballot Provisioning. 42
5.4.4 Ballot Marking 43
5.4.5 Ballot Submission 44
5.4.6 Ballot Verification and Finalization 44
Chapter 6 Op-Ed Voting Security Evaluation 46
6.1 Assumptions 46
6.2 Analysis 46
6.2.1 R1-Ballot Primary Compromise. 46
6.2.2 R2-Enable Voter Coercion and R3-Enable Vote Buying. . . . 47
6.2.3 R4-Ballot Misrepresentation. 47
6.2.4 R5 and R6-Voted Ballot Corruption/Alteration. 47

6.2.5 R7-Crytographic Compromise 48

6.2.6 R8-Identity Misuse 48

6.2.7 R9-Malicious Payload Injection 48

Chapter 7 Conclusions and Future Research Directions 49

REFERENCES 50

LIST OF PUBLICATIONS FROM THE THESIS 56

Publication 57

Vita 57

FIGURES

Figure 2.1 Decomposition of Secret Ballot goal. 6

Figure 2.2 Decomposition of One Person One Vote goal. 8

Figure 2.3 Decomposition of Universal Access goal. 8

Figure 2.4 Decomposition of Transparent and Auditable goal. . . 9

Figure 2.5 Secret Ballot Elections Goal Conflict Graph. 11

Figure 3.1 General Voting System Reference Architecture 24

Figure 4.1 Open Voting Client Reference Architecture 27

Figure 4.2 Voting Client Use Case Diagram 28

Figure 5.1 Op-Ed Voting Concept 33

TABLES

Table 4.1 Open Voting Client Architecture Security Risks 32

ABSTRACT

Building voting systems for secret ballot elections has many challenges and is the subject of significant academic research efforts. These challenges come from conflicting requirements. In this paper, we introduce a novel architectural approach to voting system construction that may help satisfy conflicting requirements and increase voter satisfaction. Our design, called Open Voting Client Architecture, defines a voting system architectural approach that can harness the power of individualized voting clients. In this work, we contribute a voting system reference architecture to depict the current voting system construction and then use it to define Open Voting Client Architecture. We then detail a specific implementation called Op-Ed Voting to evaluate the security of Open Voting Client Architecture systems. We show that Op-Ed Voting, using voters' personal devices in an end-to-end verifiable protocol, can potentially improve usability and accessibility for voters while also satisfying security requirements for electronic voting.

CHAPTER 1

INTRODUCTION

Building technology solutions for secret ballot elections is an endeavour full of intrigue and challenges. While technology has enhanced our ability to communicate and perform everyday transactions, the use of technology in voting has been met with trepidation: Why is it so difficult to engineer a solution for secret ballot elections? One potential answer lies in the number and severity of conflicting requirements [49]. While some of these conflicts are addressed in current solutions, some of the more challenging conflicts remain largely unsolved. The conflict between individual verifiability and accessibility, for example, remains an area where significant trade-offs are necessary. Current approaches to voting system construction attempt to solve the conflicting requirements with sophisticated engineering and complicated multi-channel voting [37]. The difficulty and cost of this approach are compounded by heavy regulatory demands which tightly control system changes leading to expensive and complicated solutions [40]. In this work, we argue that this approach is fundamentally flawed and that a more open, flexible approach is necessary to satisfy conflicting requirements. We call our approach *Open Voting Client Architectures*. We believe Open Voting Client Architectures may be the solution to increasing flexibility, reducing complexity, and reducing cost of voting system construction. This architectural approach can improve the voter experience by allowing the voter to use the voting client she is most comfortable with. Voting clients will ideally be tailored to various needs such as voters with visual impairment, cognitive disabilities, color-blindness, dexterity limits, and others. In this work, our goal is to define this architectural approach and assess its security implications with the hope of inspiring others to look at ways we can innovate through more open designs. Due to the inherent risks of bringing outside technology into the voting processes, we submit that all Open Voting Client Architecture implementations are insecure except those specific implementations that are proven secure. To illustrate this architectural approach and assess its security, we designed a new voting system called Op-Ed Voting, short for **O**pen in-**P**erson **E**lectronic **D**evice Voting. Op-Ed supports open voting clients in a secure way by extending an established end-to-end verifiable voting protocol and controlling the physical environment of the voting operation.

Organization In Chapter 2, we describe voting system requirements and provide a model of their conflicts. In Chapter 3, we present our voting system reference architecture and define the components we will use to define Open Voting Client Architecture. In Section 4, we define Open Voting Client Architecture and present its security risks. In Section 5, we present Op-Ed as a sample Open Voting Client Architecture implementation. In Section 6, we provide a security evaluation of Op-Ed. In Section 7, we discuss future work and conclude this work.

CHAPTER 2

Voting System Requirements and Conflict Model

Secret ballot elections are often performed today with electronic voting systems. An electronic voting (e-voting) system is a system for casting, collecting, and reporting voter intent using predominately electronic means. The Council of Europe defines e-voting as "the use of electronic means to cast and/or count the vote"[34]. While building a computer system to collect votes seems simple at first, creating a system to run binding elections under a rigid set of technical specifications and complex legal frameworks is riddled with difficulties due to conflicting requirements. A system developer or researcher who begins without a full understanding of the goals and requirements for an e-voting system is likely to become frustrated by rework and unexpected sacrifices. A person with a better understanding the requirements is likewise likely to become frustrated by the overwhelming number of conflicts and a lack of a natural starting spot. This is where conflict identification and modeling can help. For our conflict identification and modeling effort, we have chosen to use a goal-oriented approach [44]. Managing conflicts at the goal level provides more freedom for an implementer to find adequate ways to handle conflicts, such as alternative goal refinements and operationalizations, which may result in different system proposals [45]. By applying a goal-oriented approach to identifying e-voting requirement conflicts, this work hopes to enable eventual conflict analysis and full conflict resolution leading to better and more robust electronic voting solutions for secret ballot elections.

Electronic voting can refer to various implementation models ranging in the level of assistance and reliance on electronic components. [46] identifies at least eight different elections forms ranging from paper-based electronic voting systems to remote electronic voting. In nearly all cases, electronic voting systems are systems of systems that work together to define the election, design the ballots or voting interface, deliver ballots, collect voter intent, tabulate and report results. Moreover, a jurisdiction may deploy more than one electronic voting system or merge various implementations to meet the demand of their diverse voting population. This creates complex interactions which are difficult to evaluate without a proper identification of the high-level goals for an electronic voting system. Relying upon related work in requirements engineering, this work presents four primary goals as the basis for its analysis. These goals are universal for any secret ballot election and must be met by any viable electronic voting system. These are:

- Voters are afforded a **Secret Ballot**
- **One Person** is afforded and limited to **One Vote**
- Voters are provided **Universal Access** to the voting process
- The voting process is **Transparent and Auditable**

Further descriptions for these goals and a mapping to Volkimer's principles [47] are provided in 2.2.

2.1 Our Approach

Our approach builds upon these main goals to identify sub-goals which we then use to identify and classify conflicts. We selected a goal-oriented approach because it provides flexibility for implementation while showing the natural tension between requirements. According to [44], goals are recognized to provide the roots for detecting conflicts among requirements and for eventually resolving them. Modeling the interaction of goals supports a requirements elaboration process that is more accurate and more likely to yield a viable product.

We selected a manual approach to conflict identification, as opposed to automated techniques. A manual approach is where conflicts are identified by a requirements engineer or subject matter expert, which the author is both. An automated approach applies conflict identification techniques using software tools. Automated techniques typically require the use of formal specifications for requirements and any mistake that occurs during the formalization may lead to incorrect conflict detection. Our specific conflict identification approach analyzes the goals as implemented in real and theoretical e-voting applications and identifies examples of where trade-offs were necessary due to goal conflicts.

Finally, we classify conflicts as either interference or divergence. Interference is defined in [28] as the negative contribution of one requirement on another. Interference causes tradeoffs in satisfying a set of requirements and often means the requirements cannot be satisfied at the same time. Divergence between requirements refers to situations where some combination of circumstances can be found that makes the goals/requirements conflicting [45]. Divergence is a frequent, weaker form of conflict [45].

Managing conflicts is a requirements engineering activity that consists of three main activities: conflict identification, conflict analysis, and conflict resolution. Conflict identification detects the potential conflict. Conflict analysis evaluates and investigates potential conflicts and their tradeoffs. Conflict resolution resolves the potential conflict [28]. Our effort focuses on conflict identification with some discussion of conflict analysis. Conflict identification is visualized using a goal modeling graph. This model can be used to evaluate current implementations, to assist in making trade-off decisions for future implementations, to assist researchers in conducting more focused conflict analysis, and to identify the primary areas in

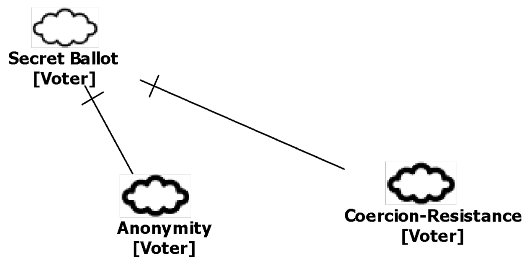

Figure 2.1: Decomposition of Secret Ballot goal.

need of conflict resolution. Future research is proposed for conflict analysis and conflict resolution.

2.2 Electronic Voting Goals

In this section, we elaborate on our 4 primary goals and how they decompose into sub-goals (not soft goals).

2.2.1 Secret Ballot Goal

A Secret Ballot election is fundamental to a voting process where voters feel they have the freedom to vote their true conviction. This is achieved by giving voters confidence the process will not divulge their vote and by preventing voter coercion. Our Secret Ballot goal maps to the Secret and Free principles from Volkamer [47]. The Secret Ballot goal is decomposed into the following two sub-goals (see Figure 2.1):

Voter Anonymity Voter's selections must only ever be known to the voter and anyone he/she willingly shares them with. There must be no means for anyone to obtain the identity of the voter who cast an individual vote or ballot, now and in the future. The concept of everlasting privacy was expressed by Moran and Noar in [32] to define the future portion of this requirement.

Coercion-Resistance The system must protect against vote selling and voter coercion. This is achieved by allowing a given voter to cast their ballot as they truly wish even in the event someone is coercing them to vote a certain way. Voters must also not be able to prove how they voted to anyone even if they wish to do so [7]. Systems achieving coercion-resistance often do so with the property of *receipt-freeness*[8]. This property requires that voters not be allowed to retain possession of anything that can be used as proof to another person of how she voted.

2.2.2 One Person One Vote Goal

In democratic elections, each voter's vote has equal weight with every other voter's vote. This is represented in the Equal principal in Volkamer's work [47]. This is achieved by treating all votes equally and by ensuring a voter only cast one valid ballot. This is often referred to as the *One Person One Vote* concept. There are some special elections where entities, such as corporations and property owners, may be given multiple votes. Thus, this goal could alternatively be written as "one person has the number of votes specified by law". The goal for One Person One Vote is achieved by satisfying the following two sub-goals (see Figure 2.2):

Voter Authenticity The voting process must only be utilized by legitimate and authenticated voters. The identity of the person must be established with proof that the person is who they claim to be.

Ballot Accountability Known in [47] as the Direct principle, the system must record who has received a ballot and prevent attempts to introduce more than one binding ballot per voter. The system must provide a means to audit the number of binding ballots compared to the number of authenticated voters.

2.2.3 Universal Access Goal

Giving all voters who wish to vote an equal opportunity to vote is critical to a fair and legitimate election. This is achieved by executing a voting process which

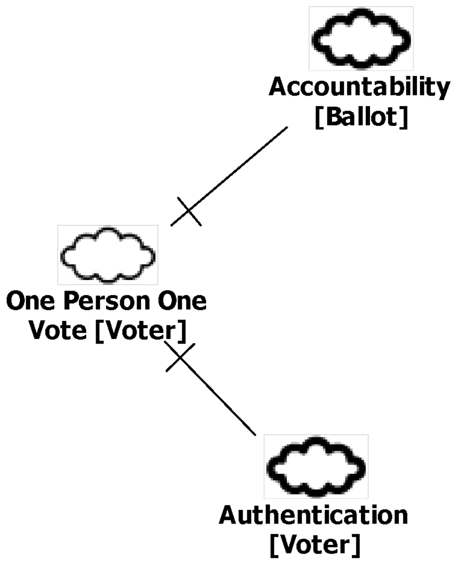

Figure 2.2: Decomposition of One Person One Vote goal.

does not introduce any undue burden on any voter. This concept of Universal Access is codified in various international laws and incorporated by Volkamer as the Universal principal [47]. The following sub-goals are required for Universal Access (see Figure 2.3):

Voter Usability The voting process must be simple and intuitive for voters of various cognitive abilities and cultural sensitivity [34]. Voters must be able to negotiate the process effectively, efficiently, and comfortably [17].

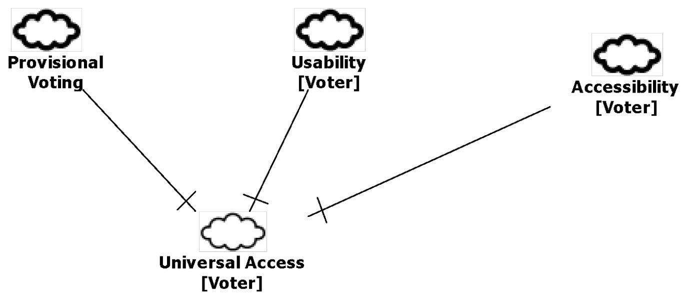

Figure 2.3: Decomposition of Universal Access goal.

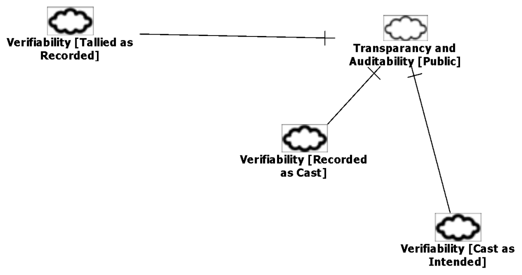
Verifiability [Tallied as Recorded]
Transparancy and Auditability [Public]
Verifiability [Recorded as Cast]
Verifiability [Cast as Intended]

Figure 2.4: Decomposition of Transparent and Auditable goal.

Voter Accessibility The voting process must allow voters with various physical impairments to vote[34]. Accessibility is measured by the degree to which the system is available to, and usable by, individuals with disabilities [17]. The most common disabilities include those associated with vision, hearing and mobility, as well as cognitive disabilities [17].

Provisional Voting Voters with questionable authenticity or eligibility at the time of voting should be allowed to cast their ballots and prove their eligibility later. Once voted, such ballots must be kept separate from other ballots and are not included in the tabulation until after the voter's eligibility is confirmed [17]. This type of voting is not deployed everywhere.

2.2.4 Transparent and Auditable Goal

Transparency and auditability ensure that the public can verify the process was accurate and reliable. This is covered by Volkamer with her Trust principle [47]. Our goal covers the actual accuracy and reliability in the voting process as well as the ability to prove the process was accurate and reliable. This leads to correct results which are accepted by all parties, including and especially the party which lost. To achieve this goal, we lean on end-to-end verifiability principles from [36] which are included as sub-goals below (see Figure 2.4):

Cast as Intended Verifiability The voter must be provided the opportunity to verify the voting system correctly interpreted her selections on the ballot [36]. This verifiability is individual and must be done while the voter can still revoke or choose not to cast her ballot.

Recorded as Cast Verifiability The voter must be provided the opportunity to verify that her vote or ballot was received and correctly recorded by the voting system [36]. In addition, the public must be able to verify that each recorded ballot is subject to the recorded as cast check.

Tallied as Recorded Verifiability The public must have the option to verify the vote was correctly tabulated from the same set of ballots which were cast by voters, and that only ballots from eligible voters were included in the final tally [36]. Verifying that the same set of ballots subject to the recorded as cast check is the same as the set of ballots subject to the tallied as record check is referred to as Consistency by Popoveniuc, et al. We incorporate this concept into the Tallied as Recorded verifiability goal for simplicity.

The 3 sub-goals provided above are considered the minimum requirements for an end-to-end verifiable voting system. End-to-end verifiable voting systems are one type of software-independent voting system discussed by Rivest and Wack in [38]. A voting system is software-independent if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome [38].

2.3 Electronic Voting Goal Conflict Model

Identification of goal conflicts was performed by literature review, review of e-voting implementations, and reasoning with the definitions provided in the previous section. In this section, we discuss each conflict by providing our rationale and citing implementation examples which demonstrate the conflict. It is important to mention that conflict identification is an ever-changing task. As new implementa-

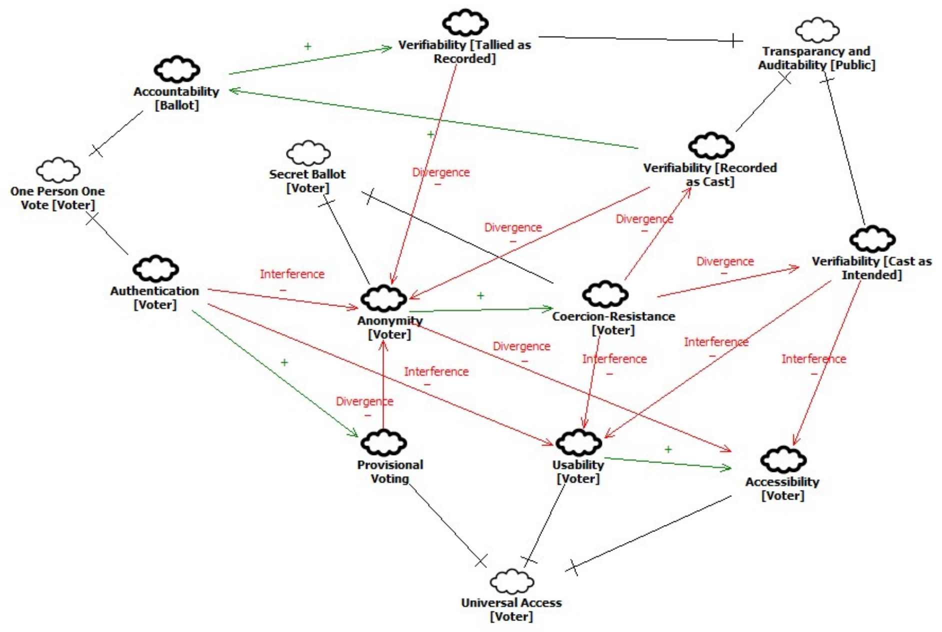
Verifiability [Tallied as Recorded]
Transparancy and Auditability [Public]
Accountability [Ballot]
Verifiability [Recorded as Cast]
Secret Ballot [Voter]
One Person One Vote [Voter]
Verifiability [Cast as Intended]
Authentication [Voter]
Anonymity [Voter]
Coercion-Resistance [Voter]
Provisional Voting
Usability [Voter]
Accessibility [Voter]
Universal Access [Voter]
Interference
Divergence
+
−

Figure 2.5: Secret Ballot Elections Goal Conflict Graph.

tion ideas surface, the potential for additional conflicts may exist.

Drawing from goal modeling [44], the graph in Figure 2.5 depicts the main goals, their sub-goals, and the edges between sub-goals. Edges between sub-goals capture the conflict classification as Interference or Divergence. Conflicts (negative relationship) are represented by minus (-) signs and red color. Some positive relationships are depicted with the plus (+) sign. Discussion of positive relationships is outside the scope of this work.

2.3.1 Voter Authentication Interferes with Voter Anonymity

Voter anonymity is best accomplished in a system where the identity of the voter is never introduced for any purpose. Thus, the goal of voter anonymity conflicts with voter authentication since voter authentication requires the identity of the voter be known and proven to the system.

Direct or indirect links between the voter's selections and the voter's identity could lead to a compromise of the voter's anonymity. Any voter authentication process will require a record of the voter along with other attributes necessary to authenticate the person as that voter. This information, along with metadata such as the time of the authentication, is ripe for intentional or unintentional misuse and connection with the voter's selections. This is especially true when voting system artifacts are retained for many months or years depending on local law. The longer the artifacts are retained, the more likely the procedural or technological mitigations put in place to limit the interaction of voter identities and voter selections are to be circumvented.

The most common resolution for mitigating this conflict in electronic voting systems is separation of responsibilities. In this approach, the capturing of voting selections is done in a system which has no knowledge of the voter's identity. The voter authentication is done on a separate system and there is no connection between the systems. This is achievable in scenarios such as in-person voting where procedural controls ensure that only authenticated voters can use the vote capture component of the system. In other scenarios, such as absentee balloting or internet voting, separation of responsibilities is much more difficult, if even possible. In those scenarios, weaker resolutions are deployed such as the two-envelope system used in postal voting. In the two-envelope system, the ballot is inserted inside an inner envelope which is then inserted in an outer envelope. The outer envelope is used for postal delivery. The voter is authenticated using his signature on the inner envelope. Authenticated voters' ballots are removed from the envelope and separated to preserve voter privacy. Digital implementations encrypt and digitally sign the voter's selections and don't allow them to be decrypted in the same system or process where the digital signature is still associated. These systems use decryption mixnets or homomorphic encryption to ensure the plain text vote and voter identities are not associated [4]. Since even the best of these mitigations rely on procedural controls and lack of collusion to protect voter anonymity, we classify this conflict as interference.

2.3.2 Voter Authentication Interferes with Voter Usability

Determining the best and most usable way to evaluate a voter's identity and eligibility to vote is a matter of intense political and technical debate. The debate centers around the balance of permitting every eligible voter the ability to vote and with requiring a certain level of proof. The more proof required, the lower the usability and the more likely eligible voters are to be incorrectly rejected. Since nearly all voter authentication implementations hurt voter usability to some level, this conflict is classified as interference.

Voter authentication can take many forms. The most common form of voter authentication in American election processes are polling place check-ins with government identification and signature-based authentication for absentee ballots. In remote electronic voting systems, voters use digital forms of authentication such as username/password, requests for personal identifiable information (name, government issued ID numbers, date of birth), smart cards, etc.

Authentication and usability are both essential in the voting process. However, access control requirements and adequate usability are frequently in conflict with each other [10]. According to [10]'s comparative analysis, the authentication methods which achieve the highest security rating only achieve a moderate usability rating at most. Electronic voting does not present any unique aspects to this conflict, so we intentionally abbreviate the discussion of this conflict.

2.3.3 Voter Anonymity Diverges with Voter Accessibility

The 2002 Help America Vote Act (HAVA) requires all American election jurisdictions to provide the same opportunity for access and participation, including privacy and independence, equally to all voters [1]. Providing privacy is difficult while providing universal access to voters with various disabilities. For instance, one of the easiest ways to provide voters with disabilities a full voting experience is to provide qualified help. However, doing this will violate that voter's right to a secret ballot. As discussed in [4], adapting voting systems to provide audio/visual aids or human assistance for voters with disabilities may create situations where the voter's candidate choice is revealed to a third party. Likewise, remote voting systems, such as postal or internet voting, are significantly more accessible compared to in-person voting at polling stations but result in a marked deterioration in voter anonymity. We classify this conflict as divergence because the techniques discussed do reduce the risk to voter anonymity to a level accepted by most voters with accessibility needs.

While outside of the scope of this work, not all requirement conflicts are pairwise. In reality, there are many complex interactions that occur when 3 or more requirements are combined. One example is the interaction between voter anonymity, voter accessibility, and voter verifiability. For example, Votegrity [14] was one of the very first voter verifiable, privacy preserving end-to-end verifiable voting schemes. In this scheme, Chaum uses visual cryptography to split an image into multiple shares. Individual shares do not yield any meaningful information about the original image. While in the voting booth, the voter can see her vote by combining two strips of paper. The voter randomly chooses one of the strips as a receipt to take home [4]. This provides voter verification and privacy protection but is not accessible since the use of visual cryptography is not usable by persons with visual impairments. While this limitation is unique to the use of visual cryptography, non-accessible mechanisms are often used to provide voter verification in a private way. Another example is the provision of ballot receipts which can be compared to a public bulletin board later. These are often long, seemingly random, and otherwise meaningless strings which are difficult for voters with cognitive or other impairments to use.

2.3.4 Coercion Resistance Interferes with Voter Usability

Many of the methods used to achieve coercion resistance ultimately hurt voter usability. This is intuitive because voting in the most straight forward, usable manner allows a coercer the opportunity to simply observe the act of voting. This is especially true in any remote voting method, such as postal or internet voting. Only supervised in-person voting can truly provide coercion resistance without hurting voter usability. In many remote voting proposals, the approach to coercion resistance is multiple voting. These proposals allow the voter to vote multiple legitimate ballots and take the last one while other schemes allow the voter to cast fake ballots which look real to a coercer. Each of these schemes, however, present usability challenges for voters who are often confused about which ballot was cast or – worse – if they cast a real ballot at all. Therefore, we classify this conflict as interference.

JCJ and its successor Civitas, for example, are known for their high level of coercion resistance but have significant usability issues as detailed in [33]. JCJ/-Civitas allows voters to generate fake credentials which are indistinguishable from real ones and can be used to cast dummy votes. Dummy votes show up on the bulletin board for verification but are ultimately not counted. Usability is hurt with this scheme because the voter is forced to manage multiple tokens and know which one is the right one and make sure to use the right one in the free moment(s) she may have away from her coercer. The usability of JCJ/Civitas was improved by [26] by adding smart card support. If the voter enters his correct Pin, the correct ballot is created with his real credentials. Otherwise, a fake credential is used. This could create a scenario where the voter mistypes his PIN, thinks he cast his legitimate ballot, but he did not cast a ballot at all.

2.3.5 Coercion Resistance Diverges with Cast as Intended

Cast as intended verifiability is intended to prove to the voter that the system properly interpreted how they voted. The simplest way to do this is to show the voter the results of the interpretation, or tabulation. This is not possible, however, because it would give the voter proof that the she can take away and provide to a coercer. Therefore, providing cast as intended verification must be done in a more complex and less straight forward manner. It must provide proof to the voter in such a way that the voter can't take that proof to others. There are known techniques to achieve both of these goals, such as the Benaloh Challenge [8], so we classify this conflict as divergence. It is critical to mention that many of the implementations which meet both of these goals negatively impact voter usability.

A good example of this is seen in the Norwegian Internet Voting Protocol [23]. In this internet voting system, cast as intended verification is provided by means of distributing voting cards to voters and then transmitting return codes to the voters via text message. If the return codes calculated by the system and provided to the voter via text message match the values on the voting card, the voter can be assured the system interpreted her vote correctly. This practice, however, introduces concerns over vote buying as discussed in [5]. Text messages simplify the task of coercers and vote-buyers because they need only ask the voter to provide the appropriate proof generated by the internet voting system itself [5]. Supporters of this protocol point to the multi-voting support as the means by which the voter can provide proof to the coercer while casting a different, non-coerced ballot. As discussed earlier, multiple voting often leads to usability issues so while this approach may satisfice this conflict, it moves the concern to a different conflict.

2.3.6 Coercion Resistance Diverges with Recorded as Cast

As first highlighted in Benaloh's seminal work on receipt-freeness [8], cryptographic election schemes have the potential to suffer from a deficiency which allows the voter to prove to a third party how their vote was cast. Benaloh points out that this deficiency is a result of attempting to produce correctness proofs of the election tally.

Recorded as cast implementations must provide proof to the voter that her ballot was received in its correct form by the election authority. Since this operation is often not immediate, implementations of this verification typically rely on a public bulletin board mechanism. Further, this verification approach is not a simple ballot tracking verification – which would verify the system received a ballot from the voter – but a verification that proves the system received the ballot the voter cast. This distinction is important because it means the system must provide proof to the voter that the contents of the ballot are the same as when the voter cast it. Because of voter coercion concerns, this proof must not actually convey the contents of the voter's ballot. This forces the implementation to use a more complex and less straight forward mechanism to provide proof to the voter but no one else. Here again, this conflict is satisfiable with current technology so we classify it as divergence with the note that its satisfaction is often at the expense of voter usability.

One example that failed to sufficiently address this conflict was the Rijnland Internet Election System (RIES), used by about 20,000 expatriate voters in the 2006 Dutch parliamentary elections [25]. The system works by publishing a reference table before the elections, including (anonymously) for each voter the hashes of all possible votes, linking those to the candidates. The original votes are only derivable from a secret key handed to the voter. After the elections, a document with all received votes is published. This allows for two important verifications: a voter can verify his/her own vote, including the correspondence to the chosen candidate, and anyone can do an independent calculation of the result of the elections, based on this document and the reference table published before the elections [24]. The voter is provided recorded as cast verification because if the voter's vote has been registered incorrectly, or not at all, the voter can detect it. This system's fundamental flaw is that the voter verification scheme can also be used to sell votes. If the voter lets someone else verify their vote, he or she could pay the voter for making the right choice [24].

2.3.7 Provisional Voting Diverges with Voter Anonymity

Provisional voting is sometimes referred to as conditional or second-chance voting. As these names indicate, provisional voting provides voters an opportunity to vote if something goes wrong on Election Day under certain conditions. Often, the voter has appeared at the wrong precinct or forgotten a required form of identification. While the ballot is cast on Election Day, the ballot is not counted until the conditions are met. This means the ballot must be held separate from counted ballots and stay associated with the voter herself, so it can be identified. In fully electronic systems, this creates a significant risk to voter anonymity. In paper ballot systems, the voter's identity is only associated with the ballot until the eligibility is determined and then the ballot is separated from the identification. In a digital system, the ballot and voter identity may never be fully separated even though the ballot is counted. Since this concern is limited to digital systems, we classify this conflict as divergence.

One example of this conflict is blockchain voting schemes. To support provisional voting, records of the ballot must be added to the blockchain which are not completely anonymous. There must be a link – indirect or direct – back to the voter. There may be another block added to the chain later to denote the ballot's ultimate status, but the original block still contains the ballot and the voter's identity. In one blockchain implementation we are aware of and likely others, the ballot is encrypted and the voter identification is a unique ID related only to the voter through an offline system. These techniques mitigate the conflict, but they do not solve it. This conflict is also present in offline e-voting devices.

2.3.8 Cast as Intended Interferes with Voter Usability

The desire to add cast as intended verifiability to the voting process requires that voters perform some action. This action may be as simple as deciding whether to verify the ballot. As expressed in [4], requiring voters to verify their vote negatively impacts usability by adding extra steps to the process and possibly confusing the voter. Since there is no current implementation that provides voter verification without an extra, undesirable step in the voting process, this conflict is classified as interference.

One example of this conflict is in the Prét à Voter [39] scheme. The key idea behind the Prét à Voter approach is to encode the vote using a randomized candidate list. The randomization of the candidate list on each ballot form ensures the secrecy of each vote while providing one half of the ballot as a receipt for cast as intended verification. Because the scheme used in Prét à Voter alters the printing of the ballot, the voter must also have a way to verify the ballots are well-formed [36]. This check is provided by giving the voter the ability to request two ballots, one to audit and spoil and one to cast. Similarly, in the PunchScan system, the voter can detect maleficence by choosing either the top or the bottom page to keep as her receipt [36]. In both cases, the cast as intended check requires the voter to perform actions well outside the nominal voting experience. Collecting more than one ballot or choosing which receipt to keep contributes to voter confusion and detracts from usability.

It is also important to note that usability is the main factor in whether voters choose to perform the verification process. When given the choice, voters will not choose to perform the verification if it is confusing, inefficient, and seemingly unimportant. This is also reported in a usability study performed on end-to-end verifiable internet voting systems by the US Overseas Voting Foundation [43]. The less voters who choose to audit the system, the less effective the verification becomes. Therefore, conflicts such as this one not only hurt usability but also negatively impact the verification goal. This dynamic is discussed in [7] and leads to many questions about how to best perform the verification, how to track whether voters performed the verification, and whether the verification can be abstracted away from voters.

2.3.9 Cast as Intended Interferes with Voter Accessibility

Cast as intended schemes rely on generating proof to the voter that the voting system has correctly handled her ballot. Most schemes fall into one of three buckets: independent encryption and compare, Benaloh challenge, and return codes. In each of these cases, the voter is asked to compare values between the voting system and a definitive source which is often on physical media or a separate system. This is very difficult for voters with disabilities, particularly with visual or dexterity limitations. We, therefore, classify this conflict as interference.

A good example of this is code voting schemes first introduced by Chaum in [13]. Code Voting gives each voter a sheet of codes with one for each candidate. Assuming the code sheet is valid, the voter can cast a vote on an untrusted machine by entering the code corresponding to her chosen candidate and waiting to receive the correct confirmation code [7]. This scheme and its successors require voters to manage the interaction between a code sheet and the system and then confirm the confirmation codes and the code sheet. This is not possible for voters with visual impairments.

2.3.10 Recorded as Cast Diverges with Voter Anonymity

In many voting schemes, voter identities are maintained along with the ballot until late into the voting process. This is true in postal voting and many forms of internet voting. This makes it difficult to provide recorded as cast verification while maintaining voter anonymity. Recorded as cast verifiability provides voters the assurance that their ballot has reached the voting authorities without compromise or deletion. This is unique proof that the ballot fidelity remained intact but must be given is such a way that no one can determine from the proof how any voter voted. Each individual proof must be free from any evidence linking the voter to a vote and there must be no way to take the collection of proofs to determine a voter's selections.

Since several schemes have been proposed which address this conflict, we classify it as divergence. These schemes provide the voter only a confirmation code which is tied somehow to the content of the ballot. In Scantegity, ThreeBallot, and other paper ballot-based systems, the voter takes part of the ballot while the corresponding part is posted to a public bulletin board [4]. The voter can reconstruct the ballot and be assured her ballot was recorded as cast. In electronic systems like Helios, a probabilistic ballot hash or ballot encryption is posted to the public bulletin board for the voter to review [4].

2.3.11 Tallied as Recorded Diverges with Voter Anonymity

As discussed in the prior conflict, voter identities are often maintained late into the voting process to provide recorded as cast verification and to generally prove the legitimacy of the ballots in the ballot box. This is necessary to show that the ballots being tallied are in fact the same ballots which came from legitimate voters.

This creates tension for the actual tabulation process which must strip away voter identification information to preserve privacy while also producing a tabulation result which is verifiably calculated from the original set of ballots. This conflict was first addressed by Chaum using decryption mixnets [4]. These mixnets rely on multiple rounds of decryption each owned by a separate election official. Each mix can be verified but it takes all trustees to accomplish the full decryption. This provides voter anonymity and tallied as recorded verifiability, but voters must trust that all trustees are not colluding.

Several improvements have been proposed which significantly address this problem using homomorphic encryption. Homomorphic encryption is a cryptographic primitive which enables ballot tabulation while still in encrypted form. The result of a homomorphic addition on a set of cipher texts is equivalent to an addition operation performed on the set of plaintexts. Only the result of the addition operation is decrypted, thereby preserving the individual voter's privacy [4]. Scratch and Vote, VoteBox, and STAR-Vote are examples of systems which use homomorphic encryption. We, therefore, classify this conflict as divergence.

CHAPTER 3

Voting System Reference Architecture

3.1 Definition

In order to propose a new architectural framework for solving requirement conflicts, we first define the current voting system reference architecture in Figure 3.1. Our reference architecture provides a template solution for any type of voting system solution, including solutions which mix digital and analog components. Our components are generalizations of commonly found components from specific implementations. We will use this reference architecture to discuss and differentiate our proposed Open Voting Client Architecture concept.

3.2 Components

3.2.1 ElectionDesigner

The Election Designer component is responsible for capturing the data and rules for the election, including the districts, offices, questions, candidates, and options which will appear on the ballots. There is typically only one Election Designer component in an election jurisdiction's voting system.

3.2.2 BallotDesigner

The Ballot Designer component is responsible for preparing voter-ready ballot objects from the Election Definition. Voting systems will contain one Ballot Designer component per ballot type (e.g. paper ballot, electronic ballot), such a one which

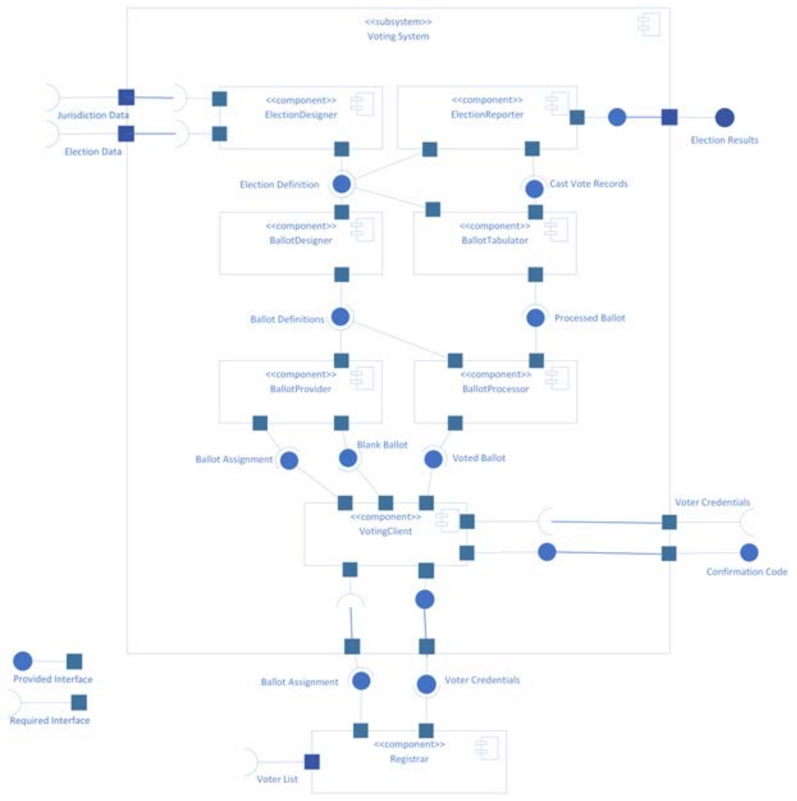

Figure 3.1: General Voting System Reference Architecture

builds PDF ballots and another which programs an electronic ballot marking device.

3.2.3 BallotProvider

The Ballot Provider component is responsible for delivering the ballot to the Voting Client component. There will be at least one Ballot Provider type per ballot type and there may be many instances of each Ballot Provider type.

3.2.4 VotingClient

The Voting Client will interact with the voter to authenticate the voter with the Registrar component and obtain the correct blank ballot from the Ballot Provider. The Voting Client component then captures voter intent, applies the intent to the blank ballot to create a voted ballot, and will submit the ballot to the Ballot Processor component. There will be at least one type of Voting Client per ballot type and there may be many instances of each Voting Client type.

3.2.5 BallotProcessor

The Ballot Processor component receives the ballot submission from the Voting Client. The Ballot Processor will validate and store the ballot.

3.2.6 BallotTabulator

The Ballot Tabulator component will process the voted ballot to create a cast vote record (i.e. a record of how the ballot is scored). There will be at least one type of Ballot Tabulator per ballot type and there may be many Ballot Tabulator instances within a voting system.

3.2.7 ElectionReporter

The Election Reporter component will receive the cast vote records, aggregate them, and generate results reports.

3.2.8 Registrar

The Registrar component is responsible for managing the list of eligible voters, performing voter authentication, and providing the Voting Client the information necessary to obtain the correct ballot from the Ballot Provider. Depending on the implementation, the Registrar component may or may not be included in the voting system.

CHAPTER 4

Open Voting Client Architecture

The goal of the Open Voting Client Architecture is to help solve several requirement conflicts by allowing an an open-market of voting clients to fully and safely interact with the voting system. This approach removes the Voting Client component from the voting system and places it outside of the voting system trust boundary. This has the significant potential benefit of engaging voters in ways specific to their needs and preferences. Among other benefits, we believe it will alleviate the conflicts of

- Voter Authentication Interferes with Voter Usability
- Cast as Intended Verifiability Interferes with Voter Usability
- Cast as Intended Verifiability Interferes with Voter Accessibility

4.1 Definitions

Let us first offer three formal definitions of Open Client Voting Architecture.

Open Voting Client Architecture A system employs an Open Voting Client Architecture if it implements all of the following publicly defined and exposed interfaces:

1. iRegistrar component for submitting the voter's credentials and obtaining the voter's ballot assignment.
2. iBallotProvider component for sending the ballot assignment and obtaining the blank ballot.

3. iBallotProcessor component for accepting voted ballots.

Permissioned Voting Client Architecture A system described by Open Voting Client Architecture which authenticates the voting clients to limit which clients can interact with the voting system.

End-to-end Verifiable Open Voting Client Architecture A system described by Open Voting Client Architecture with end-to-end verifiability properties. See [36] for a review of end-to-end verifiability properties.

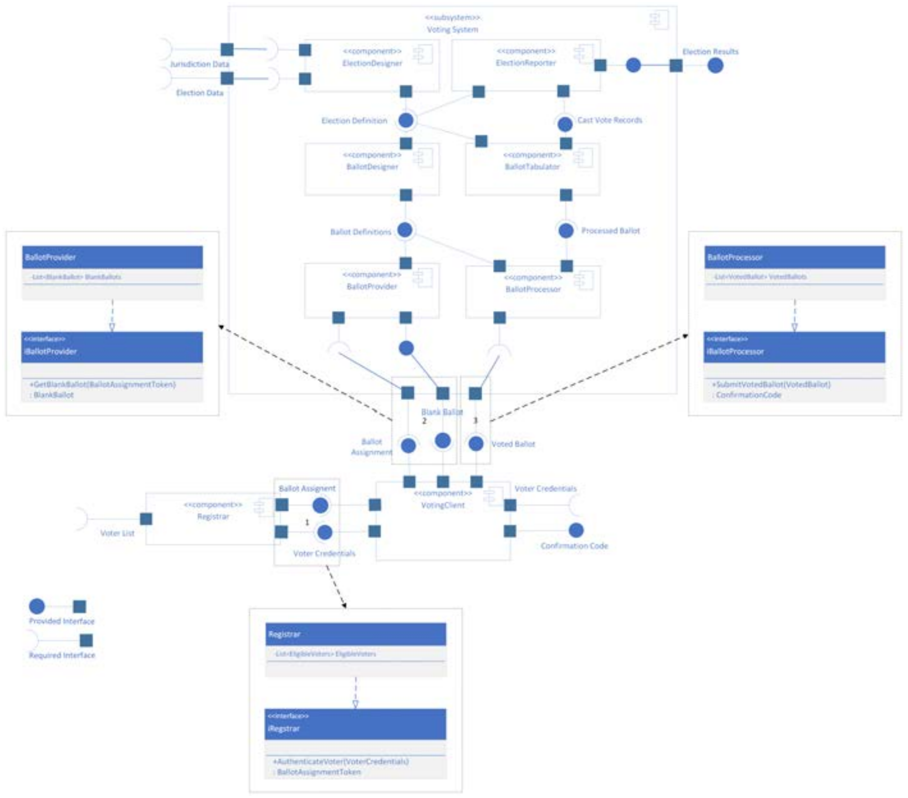

Figure 4.1: Open Voting Client Reference Architecture

4.2 Description

In order to support an open marketplace of voting clients, the voting system must provide interfaces that can be consumed by various voting client implementations. As shown in the use case diagram in Fig. 4.2, voting clients support three use cases which depend on other components: Authenticate, Obtain Blank Ballot, and Submit Ballot. A voting system with an Open Voting Client Architecture must expose interfaces to support these use cases. We define these interfaces as iRegister, iBallotProvider, and iBallotProcessor. We have defined these interfaces to be a simple as possible to ease implementation burden and make our architecture design technology-agnostic.

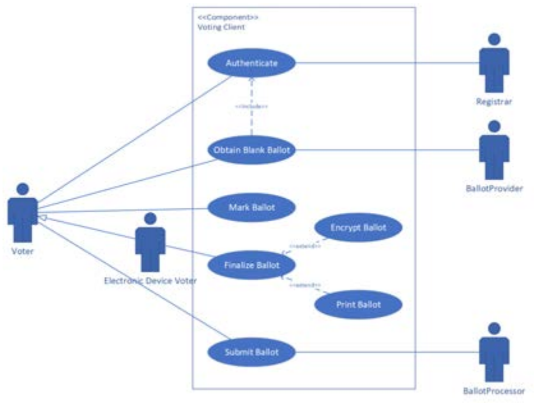

Figure 4.2: Voting Client Use Case Diagram

4.2.1 iRegistrar

The iRegistrar interface defines a single function to *AuthenticateVoter* which must be implemented by the voting system's Registrar component. The *Authenticat-eVoter* function accepts the voter's credentials and returns the voter's ballot assignment in a *BallotAssignmentToken*, if successful. The token identifies the ballot assignment and conveys the voter's right to obtain that ballot.

4.2.2 iBallotProvider

The iBallotProvider interface defines a single function to *GetBlankBallot* which must be implemented by the voting system's BallotProvider component. The *GetBlankBallot* function accepts the *BallotAssignmentToken* and returns the *Bal-*

lotBallot, if successful.

4.2.3 iBallotProcessor

The iBallotProcessor interface defines a single function to *SubmitVotedBallot* which must be implemented by the voting system's BallotProcessor component. The *SubmitVotedBallot* accepts the *VotedBallot* from the Voting Client and returns a *ConfirmationCode*.

4.3 Architectural Comparisons

We completed an architectural analysis of six voting systems, three paper based systems and three electronic voting systems, to determine how well current systems conform to Open Voting Client Architecture.

4.3.1 Paper Ballot Voting Systems

First, we examined the paper based voting systems EVS 6.0.0.0 [18], Democracy Suite 5.5 [48], and VSAP Tally 2.1 [19]. In all three systems, only one of the three required interfaces is provided. Due to their use of ballot papers, the paper systems implement an iBallotProcessor interface that can be used by various voting clients. Examples of voting clients include hand marking paper ballots, polling place electronic ballot marking devices, and absentee electronic ballot marking systems. However, we did not identify the presence of iRegistrar or iBallotProvider interfaces in any the paper ballot voting systems we examined. We found that voter authentication and blank ballot provisioning are only available in proprietary or incomplete public formats.

4.3.2 Electronic Voting Systems

Second, we examined remote electronic voting systems Helios [2], Swiss Post [42], and Voatz [41]. We found the Helios voting system provides all three required interfaces, but authenticates the voting client so is therefore an example of a *Permissioned Voting Client Architecture*. Helios implements the iBallotProvider interface by exposing an REST API to obtain the election definition. The election definition is a JSON file whose specification is available on the Helios github site [3]. Helios also implements the iBallotProcessor interface by providing a *cast* API that accepts the encrypted ballot as a JSON file. Finally, Helios implements an iRegistrar interface using OAuth which requires pre-shared client IDs and client secrets to authenticate the voting client. Helios voting client, *heliosbooth* is a Javascript client that could be replaced with different client, if not for the pre-shared client values. We also evaluated the Swiss Post and Voatz remote voting solutions. Both of these systems separate the voting client into a unique component and offer interfaces for authentication, ballot provisioning, and ballot submission. However, all three interfaces are proprietary and their public documentation is limited or non-existent.

4.4 Benefits

In addition to alleviating several requirement conflicts, this new architectural approach can have far reaching implications on voting system construction and voter satisfaction. We discuss the benefits here as potential, but unproven, since our focus is on the security analysis of this type of architecture.

4.4.1 Improved Usability

Voters will be able to choose a voting client that provides the most usable interface based on their preferences. Usability is different for each voter and involves their past experiences, expectations, and capabilities.

4.4.2 Increased Comfort/Familiarity

Voters will be able to choose a voting client that runs on their personal device (i.e. mobile phone). This will increase the voter's comfort and familiarity. Depending on how well the interfaces become standardized (something we discuss in Next Steps), the voters will likely be able to use the same voting client if they move to a different jurisdiction.

4.4.3 Flexible Accessibility

We anticipate voting clients will be built to serve voters with various disabilities: blindness, color blindness, lack of dexterity, and others. These clients can offer an experience that meets the specific needs of voters with those specific disabilities, as opposed to current solutions that generalize disabilities.

4.4.4 Reduced Costs

It is possible that the cost to build and maintain voting systems will be reduced over time as heavily engineering voting clients are replaced by open voting clients. The majority of engineering and testing costs are spent on the voting clients because they currently have to service a wide variety of voters. By removing the voting client, the overall engineering and testing cost of the voting system will be reduced.

#	Risk	Description
R1	Ballot Privacy Compromise	The voter's ballot privacy is compromised by a malicious voting client
R2	Enable Voter Coercion	Coercer controlled voting clients can force or observe voting behavior
R3	Enable Vote Buying	Buyer in control of voting client can verify the vote
R4	Ballot Misrepresentation	Voting client presents an incomplete or different ballot to the voter
R5	Voted Ballot Corruption	Voting client submits malformed ballot
R6	Voted Ballot Alteration	Voting client discards ballot or submits an altered ballot
R7	Cryptographic Compromise	Voting client compromises keys or randomness
R8	Identity Misuse	Voting client steals voter credentials and reuses to cast more ballots
R9	Malicious Payload Injection	Voting client injects malware or commands through interfaces with voting system

Table 4.1: Open Voting Client Architecture Security Risks

4.5 Security Risks

Given the shift in the trust boundary, security risks are the paramount concern with Open Voting Client Architecture implementations. Using the threat trees from [35], we derived the a list of risks that are introduced or exaggerated by the shift in voting client construction in Table 1. We specifically choose an internet voting threat model because internet voting shares the similarity that voting clients can not be trusted. We later use this list to evaluate the security of our proposed Op-Ed voting system.

CHAPTER 5

Op-Ed Voting Implementation

Our goal with Op-Ed voting is to create a voting system that meets the *End-to-end Verifiable Open Voting Client Architecture* definition in a simple and secure way. Our hypothesis is that we can utilize and extend an existing end-to-end verifiable protocol in our open voting client architecture to mitigate the risks identified in Table 4.1. We believe our implementation shows the potential for this architectural approach to improve universal voting access voting and increase voter satisfaction.

The Op-Ed, Open in-Person Electronic Device, voting system allows voters to bring their own voting clients (e.g. mobile phones, tablets) into a polling location to cast their vote. See Fig. 5.1 for a conceptual diagram.

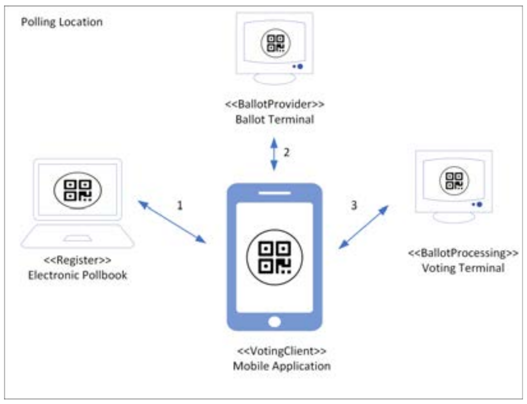

Figure 5.1: Op-Ed Voting Concept

5.1 End-to-end Verifiable Protocol Selection

We evaluated two published end-to-end verifiable protocols for use in Op-Ed: Guasch-Neuchâtel protocol [12, 22] and Microsoft's ElectionGuard [6]. Both these protocols have robust implementations, have been used to mitigate the risks of untrustworthy voting clients, and are usable in a polling place context.

In our evaluation, we looked at three aspects to determine the most suitable protocol: usability, complexity, and extensibility.

Usability. On usability, the main difference is in their approach to voter verifiability. The Guasch-Neuchâtel protocol uses a code-based approach, while ElectionGuard uses a challenge-based approach [27]. Studies like [29], consistently find code-based approach more usable than challenge-based.

Complexity. On complexity, we found that the Guasch-Neuchâtel protocol is significantly more complex than ElectionGuard. Guasch-Neuchâtel [22] defines nine algorithms for the voting scheme: Setup, Register, Vote, ProcessBallot, RCGen, RCVerif, Confirm, FCGen, Tally, and Verify - seven of these are used during the act of voting. The act of voting with ElectionGuard requires their Encryption algorithm and their Tracking Code Generation algorithm. ElectionGuard's simplicity reduces the information communicated to and from the voting client, which will yield simpler interfaces and less complex voting clients.

Extensibility. On extensibility, we found that ElectionGuard offers an advantage here due to its open source license. Whereas, the Guasch-Neuchâtel protocol has been implemented by a private company and is harder to reuse and extend.

For the above reasons, we selected ElectionGuard for Op-Ed voting. But we believe open voting client architectures can also be built using Guasch-Neuchâtel protocol.

5.2 About ElectionGuard

ElectionGuard is an open source software development kit (SDK) that makes voting more secure, transparent and accessible, according to Microsoft[11]. ElectionGuard enables end-to-end verification (E2E-V) of elections. Microsoft licenses ElectionGuard under an MIT License [31] which enables voting systems to incorporate ElectionGuard SDK into their products.

To achieve end-to-end verifiability, the ElectionGuard SDK leverages homomorphic encryption. In 1985, Benaloh first applied homomorphic encryption to electronic voting [16] and he is the lead cryptographer for ElectionGuard. Homomophic encryption allows ballots to be tabulated while still in encrypted form. Using an encryption algorithm with homomoprhic properties, the result of the addition of ciphertext ballots is equivalent to an addition operation performed on the set of plaintexts. Voter privacy is maintained by only decrypting the result of the addition operation [4]. Benaloh, and others since, use partially homomorphic encryption (PHE). PHE only allows one type of operation, such as addition or multiplication, but not both. PHE algorithms, like ElGamal and Pallier, have been widely used for this purpose.

With ElGamal encryption at its base, the Electionguard SDK uses a complex protocol to generate election parameters, generate key pairs for various guardians, generate an election key pair, encrypt ballots, aggregate ballots, and decrypt the aggregate result. With each cryptographic operation, ElectionGuard generates cryptographic proofs about election keys, ballots, and tallies using a combination of four techniques [6]: Schnorr proof [9], Chaum-Pedersen proof[15], Cramer-Damgård-Schoenmakers technique [20], and Fiat-Shamir heuristic [21].

5.3 ElectionGuard Crytographic Techniques and Key Concepts

To understand the ElectionGuard, we need to offer a basic primer on the key cryptographic techniques used. Using the techniques described below, it is possible for ElectionGuard to demonstrate that keys are properly chosen, that the ballots contain the voter's intent, that ballots are properly formed, and that decryptions match claimed values.

5.3.1 Exponential ElGamal

ElectionGuard encrypts votes using an exponential form of the ElGamal cryptosystem. First, the crypto system is established with two prime numbers p and q. p is a prime for which $p - 1 = qr$ with q being a prime that is not a divisor of integer r. A generator g of the order of q subgroup $\mathbb{Z}_p^r$ is also fixed. The private key is a random $s \in \mathbb{Z}_q$ and the public key is $K = g^s modp$. As discussed in the threshold encryption section, the actual public key used to encrypt votes is a polynomial combination of separately-generated public keys.

One of the advantages of the exponential form of ElGamal encryption is its additively homomophic property. This permits two messages M_1 and M_2 to be encrypted as (A_1, B_1) and (A_2, B_2) such that the component-wise product (A, B) is an encryption of the sum of M_1+M_2. This property is exploited in ElectionGuard by taking the individual encryptions of a single ballot option across all ballots and multiplying them together to form an encryption of the sum for that option. Since each of the individual encryptions on the ballot are either a 0 or a 1, the sum of the for that option is the tally of votes for that option.

Specifically, in ElectionGuard, a zero is encrypted as $(g^R\ mod\ p, K^R\ mod\ p)$ and a one is encrypted as $(g^R\ mod\ p, g * K^R\ mod\ p)$. The difference is the exponent of the generator g which is not shown in the encryption of a zero because $g^0 = 1$

and is shown as g in the encryption of one which is short for g^1. When these encryptions are multiplied, the result is $(g^{\sum_i R_i}\ mod\ p, g^{\sum_i V_i} * K^{\sum_i R_i}\ mod\ p)$. This is the encryption of $\sum_i V_i$ which is vote total.

The R referenced in the encryption is a nounce derived from a single 256-bit master nounce R_B for each ballot.

5.3.2 Threshold Encryption

Threshold ElGamal encryption is a form of encryption that combines individual public keys into a single public key. It also offers a homomorphic property that allows individual encrypted votes to be combined to form encrypted tallies.

This method of encryption is useful because elections are governed by an election board comprised of three or more people. Threshold encryption allows us to disperse control of decryption across the multiple people that ElectionGuard refers to as guardians. The guardians of an election will each generate a public-private key pair. The public keys will then be combined into a single election public key which is used to encrypt all selections made by voters in the election.

This approach makes the key generation process much more complex than usual. First, each of n guardians, denoted by T_i, generates an independent ElGamal public-private key pair. Each key is published in the election record along with a non-interactive zero-knowledge Schnorr proof of knowledge of the private key. These public keys are then joined into a single election public key of

$$K = \prod_{i=1}^{n} K_i\ mod\ p$$

At the end of the election, guardians are able to create partial decryptions using their individual private keys. These partial decryptions are verified and combined for the full decryption.

5.3.3 Schnorr Proof

A Schnorr proof allows the holder of an ElGamal secret key s to interactively prove possession of s without revealing s. The proof can be converted to a non-iterative proof when combined with the Fiat-Shamir heuristic discussed later.

The Schnoor proof works with discrete logarithms which is how the ElGamal public key is derived from the private key. Therefore, it can be used to be proof knowledge of the private key. First, the prover commits to randomness r by calculating $t = g^r$, where g is the group generator. The verifier then replies with a challenge c chosen at random. The prover then combines the original commitment r, the challenge c, and the private key s with $x = r + cs$. If $g^s = ty^k$, where k is the public key, is satisfied, the prover has satisfied the Schnorr proof.

In ElGamal, the Schnorr proof is used heavily during the key generation process. Each guardian generates their public-private key pair along with a random commitment and challenge value. These values are published and available for a verifier to check that the guardians did not cheat the process.

5.3.4 Chaum-Pedersen Proof

A Chaum-Pedersen proof allows an ElGamal encryption to be interactively proven to decrypt to a particular value without revealing the randomness used for encryption or the secret decryption key s[6].

In our case, the system wants to prove that the encryption is an encryption of a zero or a one. Since Chaum-Pedersen is also a proof of knowledge based on discrete logarithms, the prover commits to a random value in $\mathbb{Z}_q$ with $(a, b) = (g^u \ mod \ p, K^u \ mod \ p)$. The prover also create a pseudo-random challenge value c using the Fiat-Shamir heuristic and calculates $v = (u + cR) \ mod \ q$. A verifier can then verify the claim by checking that both $g^v \ mod \ p = a * \alpha^c \ mod \ p$ and $K^v \ mod \ p = b * \beta^c \ mod \ p$ are true. This is the basic Chaum-Pedersen proof for one value, but we need an approach that works for zero or one without revealing which one it is. This is where the Cramer-Damgård-Schoenmakers technique is used. With this approach, the prover selects a single challenge value c and must then provide challenge values c_0 and c_1 such that $c = c_0 + c_1 mod\ q$, where c_0 is the commitment to the zero value and c_1 is the commitment to the one value. Since the prover has freedom to chose one of the commitment values, the provider can fix one in advance, the prover can generate a faux claim with its chosen challenge value. The verifier can see that one of the two claims is true but cannot tell which.

Chaum-Pedersen is also used to prove that the selection limits for each contest limit for each contest has not been exceeded. This is important because it is not enough to only prove that encryptions are of zero or one. Since we don't know if it is a zero or one, a malicious actor might encrypt all ones which is an invalid vote and, therefore, we have to make sure the vote limit is not exceeded. To do this, each encrypted vote of zero or one is homomorphically combined for a contest. The Chaum-Pedersen proof is then generated to proof it is an encryption of L, the selection limit.

Finally, the Chaum-Pedersen proof is used by each guardian to prove the correctness of their partial decryption of the result. This proof establishes knowledge of the guardian's private key s for which the decrypted results, M_i, was decrypted and that it is the same private key for the public key, K_i, that was published.

5.3.5 Cramer-Damgård-Schoenmakers Technique

The Cramer-Damgård-Schoenmakers technique enables a disjunction to be interactively proven without revealing which disjunct is true. We discuss how this is used in the previous section.

5.3.6 Fiat-Shamir Heuristic

The Fiat-Shamir heuristic allows interactive proofs to be converted into non-interactive proofs. It does this by converting the interactive challenge into something that can be calculated by the prover, but is equally as random as the interactive challenge. This is done using a hash function which operates like a random oracle[30]. If the right inputs to the hash function are chosen, or defined like they are in the ElectionGuard prototype, the output of the hash function is random and uncontrollable by the prover, thus providing the same assurances of the interactive proof without the need to interact with a prover.

5.3.7 Tracking Codes

ElectionGuard produces tracking codes, sometimes referred to a verification code, to anonymously identify a ballot and allow the voter to confirm its presence in the final tally. The tracking code is formed as follows $H_i = H(H_{i-1}, D, T, B_i)$ where i in the index of the ballot, D is the voting device information, T is the date and time of the ballot encryption, and B_i is an ordered list of the individual encryption on the ballot. Voters are provided H_i as their tracking code and the entire hash chain is published as a part of the election record.

5.3.8 Challenge Ballots

When in possession of a tracking code, and not before, a voter is given the option to either cast the ballot or challenge it. Challenging a ballot is the process voters use to challenge the integrity of the voting system by forcing it to reveal what it would have submitted if the ballot were actually cast. A challenge ballot is verifiably decrypted, and the voter is given the opportunity to cast a new ballot. The decryption of the challenged ballot is published as a part of the election record so the voter can verify whether the system had encrypted the ballot and she intended it.

5.3.9 Election Record

Once voting is complete, the ElectionGuard election record is published containing all of the following artifacts.

- All cast encrypted ballots
- Proofs that all cast encrypted ballots are properly formed
- A tally ballot formed as the homomorphic aggregation of all cast ballots
- A verifiable decryption of the tally ballot
- All spoiled ballots
- Verifiable decryptions of all spoiled ballots

With this information, independent verifiers can confirm the correct operation of the election.

5.4 Op-Ed Components and Voting Workflow

5.4.1 Op-Ed Voting Clients

For Op-Ed voting, we consider that there exist a market of voting clients applications which can run on voter's mobile devices (i.e. smart phones and tablets). These voting clients may be constructed by altruist third parties, malicious third parties, or the voter herself. We assume that there will be high-quality voting clients implementations that vary in their target audience with focuses on demographics, ideology, or physical and mental impairments. Voters may choose from among these voting clients or choose to use an option provided by their election jurisdiction. We selected QR barcodes as the communication medium between the voting clients and the voting system components. This decision reduces the attack service and reduces interoperability issues. In future iterations, we would like to explore NFC as an alternative. Since Op-Ed voting system is a polling place system, we stipulate that voters must provide a form of ID external to the voting client and must be present in the polling location to cast their vote.

5.4.2 Voter Authentication

The Op-Ed Register component is an electronic pollbook (epb) and the iRegistrar interface is a barcode reader/writer. The epb, assisted by the pollworker, accepts and validates the voter's credentials. The epb generates the ballot assignment token, encodes it in a JSON string, and presents it as a QR code. The voter will use her voting client to scan the barcode and obtain the ballot assignment token. Op-Ed defines the ballot assignment token return object as:

- BallotStyleName - string value that identifies the ballot style
- BallotClaimId - nounce produced by the epb
- BallotClaimToken - HMAC-SHA256 computed from the concatenating the BallotStyleName and BallotClaimId and using a secret key, k_v, that we refer to as the VoterAuthorization key. This key is unique per polling location.

5.4.3 Ballot Provisioning

The iBallotProvider interface is implemented by the Op-Ed Ballot Terminal component and accepts the ballot assignment token, validates the BallotClaimToken is valid. If the request passes validation, the Op-Ed iBallotProvider interface returns the blank ballot corresponding to the BallotStyleName provided. Op-Ed defines this blank ballot return object as:

- BallotStyleURL - URL to download the correct ballot style

- BallotStyleHash - SHA256 hash of the BallotStyle and used by the voting client to verify it downloaded the correct ballot style
- BallotClaimId - nounce produced by the epb
- BallotSubmissionToken - HMAC-SHA256 computed from concatenating the BallotStyleHash and BallotClaimId and using a secret key, k_b, that we refer to as the BallotSubmission key. This key is unique per polling location.
- ElectionGuardParameters - ElectionGuard values of prime number p, generator g, and public key K.

We choose to provide the ballot style as a URL in order to reduce the size of the blank ballot object, making it easier to provide as a barcode from the Ballot Terminal component. In theory, the voter may also download the blank ballot prior to entering the polling location and pre-mark it in their voting client. The voter can then check the integrity of the ballot they pre-marked with the hash value and proceed immediately to casting the ballot without taking time in the polling location to cast the ballot. This approach enables many other variations such as drive through voting.

5.4.4 Ballot Marking

Once the voter has obtained the ballot, the Voting Client will verify the integrity using the BallotStyleHash provided. The Voting Client will display the ballot to the voter and allow the voter to mark the ballot. The details of how the voter interacts with the ballot will vary per implementation. Once the voter has made her selections on the Voting Client, the client will encrypt the voter's selections and generate the zero-knowledge proofs specified in the ElectionGuard protocol. This consist of a proof that the encryption associated with each option is either an encryption of a zero or an encryption of one, and a proof that the encrypted values in each contest is equal to the selection limit for that contest. After encryption of the ballot is complete, the voting client generates a tracking code. ElectionGuard defines the code is a hash of the encrypted ballot and acts as the voting client's commitment to the encrypted ballot [6].

5.4.5 Ballot Submission

To submit the ballot, the Voting Client builds a barcode with the Voted Ballot contents formatted as a JSON string. The voter will present the barcode to the Voting Terminal which implements the iBallotProcessor interface and accepts VotedBallot objects. Op-Ed defines the Voted Ballot object as:

- BallotStyleName - identifier for the ballot style
- BallotClaimId - nounce produced by the epb
- BallotSubmissionToken - provided by the BallotProvider component
- EncryptedBallot - ElectionGuard encrypted ballot
- Zero Knowledge Proofs - ElectionGuard specified zero-knowledge proofs of ballot correctness generated by Voting Client

5.4.6 Ballot Verification and Finalization

The Voting Terminal will validate that the BallotClaimId is unique and the BallotSubmissionToken is valid. Next, the terminal will validate the zero-knowledge proofs of ballot correctness. Finally, the terminal will re-compute the tracking code and display it to the voter along with a copy of the blank style contents. The voter can compare the tracking code provided by the voting client to the one displayed on the Voting Terminal to gain confidence her client submitted the same ballot. Furthermore, the voter can check the ballot contents to confirm they match what her voting client presented to her. Since the tracking code itself doesn't reveal how the voter voted, a blind voter can get assistance comparing the values or can use a barcode scanner to read the tracking code from the voting terminal. At this point, the voting terminal provides the voter the option to cancel the ballot (known as challenging in the ElectionGuard specification [6]) or finalize the submission. The voter may choose to cancel the ballot because 1. The tracking codes do not match, 2. The voter wishes to challenge her voting client's encryption, or 3. The voter submitted that ballot under the direction of a vote coercer or vote buyer. The voter may repeat the ballot submission process until the voter is comfortable finalizing the submission.

CHAPTER 6

Op-Ed Voting Security Evaluation

6.1 Assumptions

Our evaluation makes the following assumptions:

- The VoterAuthorization and BallotSubmission secret keys are unique per polling location and are kept secret.
- The Voting Clients and Voting Terminal do not collude.
- For privacy, we assume the voting client is honest. This same assumption is made by all remote voting schemes, but is improved in Op-Ed voting since voters can select their voting client.

6.2 Analysis

6.2.1 R1-Ballot Primary Compromise.

Ballot Privacy is maintaining the intention of a voter unknown to others. Other than observing the voter selections or the encryption randomness from the voting client (which we assume does not happen because we assume the voting client is honest for privacy), the only way to attack the privacy is to brute force the encryption. Since ElectionGuard uses a strong encryption algorithm, this type of attack is not feasible.

6.2.2 R2-Enable Voter Coercion and R3-Enable Vote Buying.

Voter Coercion and Vote Buying is not enabled unless the coercer/buyer is provided proof of how the voter voted. Since the voting is taking place in a supervised polling location, the coercer/buyer can not directly observe the act of voting. Furthermore, since the finalization of the ballot is performed on the Voting Terminal instead of the Voting Client, the coercer/buyer can not be convinced that what the Voting Client (which may be controlled by the coercer/buyer) observed was the final ballot the voter cast.

6.2.3 R4-Ballot Misrepresentation.

The Voting Client can only submit the ballot style assigned based on the Ballot-SubmissionToken. However, a malicious Voting Client could remove options from its display. This maleficence is detectable by the voter when the Voting Terminal displays the ballot contents of the ballot style. Diligent voters can detect if options were not shown to them.

6.2.4 R5 and R6-Voted Ballot Corruption/Alteration.

Attempts by malicious Voting Clients to submit corrupt or altered ballot will be rejected by the Voting Terminal through a combination of verifications performed by the Voting Terminal and the Voter. First, an attempt by the voting client to present a malformed ballot will be rejected when the Voting Terminal verifies the cryptographic proofs for ballot correctness. Second, an attempt to submit a different ballot other than what the Voting Client committed to in its tracking code will be caught by the voter comparing the two codes. Ultimately, the voter can verify the validity of the encryption by using the Cancel (i.e. spoil) feature and checking the ballot's decryption on the public bulletin board [6].

6.2.5 R7-Crytographic Compromise

Op-Ed voting does not rely on voting clients managing keys thus eliminating a set of attacks. However, the voting clients are responsible for generating the random values used in the encryption. Compromising the randomness will lead to ballot privacy attacks. While we assume honest voting clients for privacy protection, these attacks can be detected using the Cancel (spoil) feature. Since spoiled ballots are decrypted, the random values are revealed and corrupt voting clients could be exposed. The threat of detection mitigates the likelihood of an attack on encryption randomness.

6.2.6 R8-Identity Misuse

To prevent this risk, Op-Ed stipulates that voter authentication requires an external component to the authentication routine. For example, the Voting Client may submit partial voter credentials but the pollworker will verify a signature or picture before the electronic pollbooks responds with the ballot assignment token. This approach prevents the reuse of the voter credentials. Furthermore, the use of single-use, HMAC-SHA256 based tokens prevents forgery and replay attacks.

6.2.7 R9-Malicious Payload Injection

The selection of barcodes limits the interaction between the voting client and voting system components. This limit reduces the types of attacks to a code or command injection attack. Both of these are preventable with proper input handling, but the use of barcodes also makes these attacks uniquely detectable. Any party can observe what is being transmitted in the barcodes and malicious input can be detected.

CHAPTER 7

Conclusions and Future Research Directions

In this work, we define a new approach to voting system construction called Open Voting Client Architecture. This architectural approach uniquely allows for anyone to write voting clients and not trust the voting system manufacturers, and allows for voting clients to be written to assist voters with a variety of needs and preferences. We offer a formal definition for this architecture and discuss its benefits and security risks. We then presented a specific voting system implementation called Op-Ed Voting which meets our definition of Open Voting Client Architecture, and we evaluated how it meets electronic voting security requirements. Our security evaluation shows that it is possible to have a secure Open Voting Client Architecture while realizing all of the potential voter benefits. This was possible because we utilized and extended the existing ElectionGuard end-to-end verifiable protocol in our implementation.

Open Voting Client Architecture represents a new direction in voting system construction, one that allows for security assurances and voter flexibility. Our hope is that others are inspired to present other Open Voting Client Architecture implementations, like Op-Ed voting, for consideration. Future work to refine and prototype Op-Ed voting is planned along with work to standardize the interfaces between the Voting Client and voting system components. Investigation into the impact of new technology, such as those that use block chain, is also planned.

REFERENCES

[1] 107th Congress (2001). Help america vote act of 2002.

[2] Adida, B. Helios documentation. `https://vote.heliosvoting.org/docs`, 2021.

[3] Adida, B. Helios github. `https://github.com/benadida/helios-server`, 2021.

[4] Ali, S. T., and Murray, J. An overview of end-to-end verifiable voting systems. CoRR abs/1605.08554 (2016).

[5] Barrat, J., Chevalier, M., Goldsmith, B., Jandura, D., Turner, J., and Sharma, R. Internet voting and individual verifiability: The norwegian return codes. In Electronic Voting (2012).

[6] Benaloh, J. Electionguard specification v0.95. `https://github.com/microsoft/electionguard/releases/tag/v0.95.0`, 2021.

[7] Benaloh, J., Bernhard, M., Halderman, J. A., Rivest, R. L., Ryan, P. Y. A., Stark, P. B., Teague, V., Vora, P. L., and Wallach, D. S. Public evidence from secret ballots. CoRR abs/1707.08619 (2017).

[8] Benaloh, J., and Tuinstra, D. Receipt-free secret-ballot elections (extended abstract). In Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing (New York, NY, USA, 1994), STOC '94, ACM, pp. 544–553.

[9] Brassard, G., and Schnorr, C. Efficient identification and signatures for smart cards, vol. 435, 1990.

[10] Braz, C., and Robert, J.-M. Security and usability: The case of the user authentication methods. In Proceedings of the 18th Conference on L'Interaction Homme-Machine (New York, NY, USA, 2006), IHM '06, ACM, pp. 199–203.

[11] Carter, R. Electionguard github site, 2020.

[12] Castelló, S. G. Individual verifiability in electronic voting. PhD thesis, Universitat Politècnica de Catalunya (UPC), 2016.

[13] Chaum, D. Surevote: Technical overview. In Proceedings of the workshop on trustworthy elections (WOTE 2001) (2001).

[14] Chaum, D. Secret-ballot receipts: True voter-verifiable elections. IEEE Security Privacy 2, 1 (Jan 2004), 38–47.

[15] Chaum, D., and Pedersen, T. P. Wallet databases with observers; crypto'92, lncs 740, 1993.

[16] Cohen, J. D., and Fischer, M. J. A robust and verifiable cryptographically secure election scheme. Yale University. Department of Computer Science, 1985.

[17] Commission, E. A. 2005 voluntary voting system guidelines.

[18] Compliance, S. Es&s evs 6.0.0.0 certification test report. `https://www.eac.gov/sites/default/files/voting_system/files/ESS_EVS6000_EAC_Certification_Test_Report_v4.0.pdf`, 2018.

[19] Compliance, S. County of los angeles vsap tally 2.1 functional test report. `hhttps://votingsystems.cdn.sos.ca.gov/vendors/LAC/vsap2-1/vsap21-funct-reprt.pdf`, 2020.

[20] Cramer, R., Damgård, I., and Schoenmakers, B. Proofs of partial knowledge and simplified design of witness hiding protocols. In Annual International Cryptology Conference (1994), Springer, pp. 174–187.

[21] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In Conference on the theory and application of cryptographic techniques (1986), Springer, pp. 186–194.

[22] Galindo, D., Guasch, S., and Puiggali, J. 2015 neuchâtel's cast-as-intended verification mechanism. In International Conference on E-Voting and Identity (2015), Springer, pp. 3–18.

[23] Gjøsteen, K. The norwegian internet voting protocol. In E-Voting and Identity (Berlin, Heidelberg, 2012), A. Kiayias and H. Lipmaa, Eds., Springer Berlin Heidelberg, pp. 1–18.

[24] Jacobs, B., and Pieters, W. Electronic Voting in the Netherlands: From Early Adoption to Early Abolishment. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 121–144.

[25] Jones, D. W. Some problems with end-to-end voting. In End-to-End Voting Systems Workshop, Washington DC (2009).

[26] Juels, A., Catalano, D., and Jakobsson, M. Coercion-Resistant Electronic Elections. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 37–63.

[27] Kulyk, O., Henzel, J., Renaud, K., and Volkamer, M. Comparing "challenge-based" and "code-based" internet voting verification implementations. In Human-Computer Interaction – INTERACT 2019 (Cham, 2019), D. Lamas, F. Loizides, L. Nacke, H. Petrie, M. Winckler, and P. Zaphiris, Eds., Springer International Publishing, pp. 519–538.

[28] Mairiza, D., Zowghi, D., and Nurmuliani, N. Managing conflicts among non-functional requirements. In 12th Australian Workshop on Requirements Engineering (01 2009).

[29] Marky, K., Kulyk, O., and Volkamer, M. Comparative usability evaluation of cast-as-intended verification approaches in internet voting. SICHERHEIT 2018 (2018).

[30] Miller, J. Serving up zero-knowledge proofs, 2021.

[31] MIT. Mit license [dot] org, 2021.

[32] Moran, T., and Naor, M. Receipt-free universally-verifiable voting with everlasting privacy. In Advances in Cryptology - CRYPTO 2006 (Berlin, Heidelberg, 2006), C. Dwork, Ed., Springer Berlin Heidelberg, pp. 373–392.

[33] Neumann, S., and Volkamer, M. Civitas and the real world: Problems and solutions from a practical point of view. In 2012 Seventh International Conference on Availability, Reliability and Security (Aug 2012), pp. 180–185.

[34] of Ministers on 14 June 2017, C. Recommendation cm/ref (2017)5 of the committee of ministers to member states on standards for e-voting. `https://search.coe.int/cm/Pages/result_details.aspx?ObjectId=0900001680726f6f`.

[35] Pardue, H., Yasinsac, A., and Landry, J. Towards internet voting security: A threat tree for risk assessment. In 2010 Fifth International Conference on Risks and Security of Internet and Systems (CRiSIS) (2010), pp. 1–7.

[36] Popoveniuc, S., Kelsey, J., Regenscheid, A., and Vora, P. Performance requirements for end-to-end verifiable elections. In Proceedings of the 2010 international conference on Electronic voting technology/workshop on trustworthy elections (2010), USENIX Association, pp. 1–16.

[37] Reddy, A. V., and Raghavan, S. V. Architecture of multi channel multi database voting system. In Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government (Boston, MA, 2005), M. Funabashi and A. Grzech, Eds., Springer US, pp. 281–295.

[38] Rivest, R. L. On the notion of ”software independence” in voting systems. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 366, 1881 (2008), 3759–3767.

[39] Ryan, P. Y. A., Bismark, D., Heather, J., Schneider, S., and Xia, Z. PrÊt À voter:a voter-verifiable voting system. IEEE Transactions on Information Forensics and Security 4, 4 (Dec 2009), 662–673.

[40] School, T. W. The business of voting. `https://trustthevote.org/wp-content/uploads/2017/03/2017-whartonoset_industryreport.pdf`, 2017.

[41] Specter, M. A., Koppel, J., and Weitzner, D. The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in us federal elections. In 29th {USENIX} Security Symposium ({USENIX} Security 20) (2020), pp. 1535–1553.

[42] Stefanelli, R., and Monnat, D. M. X. A secure e-voting infrastructure. implementation by swiss post. Second In (2017), 326.

[43] U.S. Vote Foundation, G. The future of voting end-to-end verifiable internet voting usability study, 2015.

[44] van Lamsweerde, A. Goal-oriented requirements engineering: a guided tour. In Proceedings Fifth IEEE International Symposium on Requirements Engineering (Aug 2001), pp. 249–262.

[45] van Lamsweerde, A., Darimont, R., and Letier, E. Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering 24, 11 (Nov 1998), 908–926.

[46] Volkamer, M. Evaluation of electronic voting: requirements and evaluation procedures to support responsible election authorities, vol. 30. Springer Science & Business Media, 2009.

[47] Volkamer, M., and McGaley, M. Requirements and evaluation procedures for evoting. In The Second International Conference on Availability, Reliability and Security (ARES'07) (April 2007), pp. 895–902.

[48] V&V, P. Dominion voting systems democracy suite version 5.5 test report. `https://www.eac.gov/sites/default/files/voting_system/files/Dominion_Voting_Systems_D-Suite_5.5_Test_Report_Rev_A.pdf`, 2018.

[49] Wilson, A. Modeling conflicts in secret ballot elections. E-Vote-ID 2019 (2019), 171.

LIST OF PUBLICATIONS FROM THE THESIS

1. Wilson, A.Modeling conflicts in secret ballot elections. E-Vote-ID2019 (2019), 171 .

2. Wilson, A., Roy, S., A Novel Framework for Open Voting Client Architectures 7th Workshop on Advances in Secure Electronic Voting, 2021 (Submitted, Pending Acceptance)

VITA

Aaron Wilson is a security engineer, software architect, and election technology expert. He has a passion for building innovative and secure election technology and founded Enhanced Voting in 2013 with that vision. Enhanced Voting is an election technology company specializing in online balloting.

Aaron recently served as the Senior Director of Election Security for the Center for Internet Security (CIS). At CIS, Aaron led all election security best practice development efforts including the publication of A Guide for Ensuring Security in Election Technology Procurement, Security Best Practices for Non-Voting Election Technology, and Managing Cybersecurity Supply Chain Risks in Election Technology. Aaron was also the creator of the Rapid Architecture Based Election Technology Verification (RABET-V) Process and Pilot Program. Aaron spoke at the 2020 RSA Conference on securing non-voting election technology and is often quoted in security articles on ways to improve election security.

Aaron began his career testing and conducting security evaluations of voting systems for the Florida Division of Elections. Aaron has also served as the Vice President of Products and Services for Greenshades Software and the Director of Product for Clear Ballot Group, a federally certified voting system manufacturer. In 2010, he led the deployment of the first MOVE-act compliant electronic ballot delivery systems for overseas civilians and military voters in 10 states.