The E2EVIV Report

Many People

May 13, 2015

Contents

List of To Do Items

Chapter 1

Executive Summary (Joe K./Susan) (0%)

Chapter 2

Introduction (Joe K./Susan) (95%)

2.1 The E2E VIV Project

2.1.1 Situation

In March 2013, Overseas Vote Foundation's President and CEO began a discussion with a small group of experienced, election integrity technology advocates about how, if faced with having to specify an Internet voting system, they would respond. Security concerns being the primary reason for their general opposition to what they deemed inadequate efforts of others to do so to date, it was nonetheless agreed that taking on the question was crucial at the time.

Gridlock is not unique to Washington politics, it is quite well known around the topic of Internet Voting in the U.S. The scientific community, federal agencies, cyber security specialists and certain organized activists have strongly advised against exposing the ballots of the most powerful nation on earth to the seemingly endless range of cyber threats, which run rampant on today's Internet.

Nevertheless, faced with ongoing challenges to serve their constituencies in modern and efficient ways, and having experienced the everyday efficiencies of technology throughout their lives, when seeking new and improved election systems, election officials often want to consider Internet-based technologies. In the current climate of economic austerity, innovation in elections is rare. Our election officials are trapped in a technology no-man's land of ongoing support payments for outdated voting systems, compounded by no means of certifying new voting systems they would like.

Election integrity advocates cite that secure, tested, certified remote voting systems that election officials envision aren't available. The scientific community does not consider online ballot return systems secure, nor are they certified. As a result, email has become the default stopgap method for moving ballots online, although it does not provide any of the benefits that a secure, full-featured voting system would provide. Email is demonstrably weak on security, yet election officials and voters are using it regularly to transmit ballots because viable alternatives are not available. Examination of new and better ways to use technology to meet specific voting needs, for example, that of the remote overseas citizen, military and disabled is needed.

Existing vendors of Internet voting technologies, whose systems are neither tested nor certified, would like to openly market and sell their systems within the U.S. and not face the resistance of the election integrity advocates. No agreement on how to proceed, a years-long history of mediocre attempts, ongoing animosity between stakeholder parties, and a general lack of research on the current questions could well-describe the situation.

2.1.2   A Proposed Solution

Within this climate, a project proposal was written and funding provided by The Democracy Fund, a Washington D.C. based philanthropic organization whose stated objective is to "...invest in organizations working to ensure that our political system is responsive to the priorities of the American public and has the capacity to meet the greatest challenges facing our country."

A project intent on examining the future of voting and how it might be executed securely online fit neatly into the strategic purview of the fund; one that approached the question of Internet-voting from a research perspective, that sought to fill in the gaps of the many open questions plaguing the discussion was regarded by the supporting organization as a positive endeavor.

According to Joe Goldman, Director of the Democracy Fund, "The significance of this project will be in its ability to break open the conversation from its current stalemate and include all sides in a constructive project to openly examine and research what is really needed by voters and election officials, and to determine whether this form of voting can meet those needs and still guarantee security of the election. Equally important, it will identify potential tradeoffs and shortcomings that represent the diverse range of values we hold dear in our elections."

On December 19, 2013, Overseas Vote Foundation[1] (OVF), a nonpartisan, nonprofit organization dedicated to overseas and military voter participation announced the launch of the project, which was called the End-to-End Verifiable Internet Voting: Specification and Feasibility Assessment Study (E2E VIV Project). Its stated aim was to examine a form of remote voting that enables a so-called "end-to-end verifiability" (E2E) property. A unique team of experts in computer science, usability, and auditing together with a selection of local election officials from key counties around the U.S. assembled for the study.

They agreed to focus their efforts to produce a system specification and set of testing scenarios, which if they meet the requirements for security, auditability, and usability, would then be placed in the public domain. At the same time, their intent was to demonstrate that confidence in a voting system is built on a willingness to verify its security through testing and transparency.

There is an historical misunderstanding in the U.S. election community that the E2E VIV Project aimed to correct: that our country's best scientists are not against technology advancements, nor are they inherently at odds with the election officials who seek technology improvements to meet their administrative challenges. Rather, that the U.S. scientific community takes issue with unproven claims of security regarding existing systems that are not publicly tested or vetted. The study aimed to recalibrate this situation.

The group of scientific leaders on the project has often pointed out security vulnerabilities in past systems, however, in the face of the E2E VIV Project, they agreed on one thing: that if Internet Voting (IV) does happen, it should be in a system that takes advantage of end-to-end verifiability and auditability.

2.1.3   Definition

The term E2E is often used casually without precision. E2E-verifiability is considered a property of an election and for the purposes of the study, an E2E-verifiable election has two important components: first, that voters can individually check that their ballots are cast as they intend; and second, that anyone can check that all of the cast ballots have been accurately tallied.[2]

While systems of this nature have been developed in the past, none have been broadly used or successfully commercialized; the E2E VIV Project would make a concerted effort to be informed by these past efforts and build upon them as appropriate. Usability factors were also considered from the outset of the study to address the significant challenges faced by remote and disabled voters when using such systems to participate. A viable outcome of this study with respect to security, auditability, and usability is intended to enable development efforts to ensue.

---

[1] Since the start of the study, Overseas Vote Foundation has been renamed as "Overseas Vote", an initiative of U.S. Vote Foundation.

[2] Definition from Dr. Josh Benaloh, Senior Cryptographer at Microsoft Research.

For those concerned with election integrity, there is a justifiably negative reflex in response to IV: it takes all of the problems with current remote voting systems and adds all of the problems and security vulnerabilities of the Internet.

The E2E VIV Project sought to potentially make the case that use of the Internet enables and facilitates the introduction of E2E-verifiability (E2EV) and that the benefits of E2EV may be able to overcome the vulnerabilities introduced by using the Internet. No participant on this project discounted the concerns of voting over the Internet, nor did or do they view E2EV as a magic sauce that makes the Internet secure. Nevertheless they believe that E2EV warrants examination in regards to the properties it achieves. These properties are achieved even when votes are cast on untrusted devices like PCs and transmitted over an untrusted medium such as the Internet.

The E2E VIV Project does not attempt to make the Internet secure. Instead, E2EV negates many (although not all) of the risks of voting via the Internet while introducing substantial new benefits that are not found in currently deployed voting systems.

2.2 Goals and Objectives

2.2.1 Shared Goals

Election officials and scientists involved in elections **share** the overall goals: that voting systems can be proven secure, auditable, verifiable, and accessible.

This project is evidence that the needs and requests of election officials to explore optimum ways of serving remote voters are of deep concern to the scientific community, that they have a great motivation to address these questions when given a constructive opportunity to do so, and that they would like to work together, not at opposite ends, to examine the possibilities in this realm.

2.2.2 Project Goal

Our goal is to specify and define a system and its testing scenarios for an online voting method that can provide both security and confidence to voters that their selections are accurately recorded and counted. Our assertion is that E2E-verifiability negates many, although not all, of the risks of voting via the Internet while introducing substantial new benefits that are not found in currently deployed voting systems.

The project team presumes that if E2E VIV is a possible answer, or a step toward one, we will find out and we will see if it can answer the needs of many voters and election officials. If not, we will still gain specific knowledge about the shortcomings, which can be further acted upon in the future.

2.2.3 Additional Objectives

Presentation and discussion of the report with key stakeholders, integrating their feedback, and garnering their cceptance of the report (see below in Success section) are additional core objectives of the project.

2.2.4 Deliverables

The main deliverable of the E2E VIV Project is the development of a "whole product solution" specification (or simply specification for short) for a trustworthy E2E VIV election system.

We have produced a report presenting a system specification to create a secure E2E VIV system, a set of testing specifications to demonstrate the security, a set of guidelines for system usability, accessibility, and testing. Additional topics and analyses may be considered and discussed in the report, such as legal and administrative challenges, and ballot secrecy, privacy, and confidentiality.

2.2.5 A First Step

This project represents step one in an examination of whether one day this might be possible. Our current plan is to examine the potential for an E2E VIV remote voting system together with election officials, taking into close account their needs and the needs of disabled voters. If a system can one day be developed based on these principles, then we want to know. We need the answers that this project will bring before we can say whether we, or anyone, will build any new system. A viable outcome of this study with respect to security, auditability, and usability will enable development efforts to ensue.

2.2.6 Success

Beyond judging the outcome – the fact that this project takes a research and testing-based approach to a problem that has been "in stalemate mode" will result, we believe, in stimulating election development overall. The election industry is operating in a traditional paradigm with only a few vendors able to survive, albeit demand to move away from outdated, expensive, hardware-oriented solutions.

It would be considered a success to specify a system and testing for a usable, secure E2E verifiable remote voting technology, to identify its strengths and weaknesses and reasons to pursue or not pursue this approach to remote and/or disabled citizen voting.

However, from the beginning is was clear that if the project determines that the technology is weak and should not be developed, it would be a different outcome, but also one with many useful implications.

Success of the project can only be determined if the specification is one that is: 1) supported by the vast majority the expert teams, including the technical, usability, testing, and research teams; 2) endorsed by the vast majority of the advisory council, and; 3) endorsed by the major stakeholders in elections administration as represented by the project's local election officials.

Additionally, the E2E VIV Project expects to receive support and endorsement from many members of the electronic voting activism community, as represented by key members of the Election Verification Network and the Verified Voting Foundation.

The specification will be of a form with sufficient detail such that the following requirements are fulfilled:

Independent Implementation The specification must be of sufficient detail and clarity that an implementation of the election system must be possible by an independent party without extensive dialog with participants in the project.

Independent Validation It must be possible for a moderately proficient IT expert to objectively determine, in a reasonable time frame with reasonable cost, if any election system constructed which claims to fulfill the specification.

Evidence-Based Decisions Every decision made in the crafting of the specification must be objectively justifiable and the evidence for the decision must be traceable.

2.2.7 Scope

The original project was tightly limited to involve system specification and testing only. No system development was envisioned in this phase beyond mockups to help test usability. However, this changed early on in the project when Joe Kiniry of Galois, Inc. came on board to manage the project and its team.

The Galois engineers brought significant expertise to the project and set about to develop a set of rigorous engineering artifacts "demonstrators" fit for refinement into a working election system, and against which third parties can perform independent validation and verification. According to Galois, demonstrators are technical artifacts from the point of view of definition and constructions, but non-technical artifacts from the point of view of demonstration. Galois suggests that all demonstrators developed using Galois IR&D funding be:

- developed in a completely transparent and public fashion within the Galois GitHub Organization,

- cross-referenced, and thus traceable to and from, all specification aspects (from domain models to behavioral design specifications), are replicated into the E2E VIV GitHub Organization, and

- are licensed under either a mainstream Open Source license with a strong community or an alternative license tuned to the elections community.

Significantly, Galois is a leader in the process of computing on data while it remains encrypted, and in the automated generation, validation and synthesis of high assurance cryptographic solutions. They excel in multiple areas of cryptographic implementation, all of which can be applied to the challenge of developing secure and usable E2E VIV voting.

Relevance of Galois' work to the project was explicit: the aim to apply cutting edge computer science and mathematics to solve difficult technological problems was a clear definition of what was needed to solve the secure, verifiable election systems development challenge. Galois' management agreed to donate a significant portion of engineering time to the project in order to build "demonstrators" that would be used to prove the concepts of E2EV and to further examine security and usability.

2.3 People

Inherent in the E2E VIV Project was the opportunity to combine the abilities, knowledge, experience and expertise of a diverse group of technologists, computer scientists and election officials involved in election integrity together to form the overall project team. Technical, usability, testing and local election official sub-teams were formed for ease of communication. The technical team has decades of experience in E2E technology, cryptography, usability, and testing. An Advisory Council was established to broaden the communication with interested members of the election community.

Overseas Vote Foundation (OVF), as the official grantee, was responsible the overall project conception, proposal development, presentations, communications, management, team recruitment, contractual obligations, public relations, events and budgeting. Deep experience in the arena of overseas and military voting, absentee voting, community building, voter survey research, election reform and communications gave OVF a unique edge in managing the project.

Galois, Inc. provided the technical and engineering project management. Named as the technical project manager, Dr. Joseph Kiniry, working as a Principal Investigator at Galois, facilitated the communication and decision-making of the expert teams. He became the main author and editor of the report and ran all engineering projects and usability aspects of the study.

2.3.1 Team Members

Project Manager: Susan Dzieduszycka-Suinat, Overseas Vote Foundation

Lead Technical Project Manager: Dr. Joseph Kiniry, Galois

Technical Team

Dr. Josh Benaloh Senior Cryptographer, Microsoft Research

Dr. David R. Jefferson Lawrence Livermore National Laboratory

Dr. Doug W. Jones Associate Professor, Department of Computer Science, University of Iowa

Dr. Aggelos Kiayias Associate Professor, Computer Science and Engineering, University of Connecticut

Dr. Olivier Pereira Professor, Institute of Information and Communication Technologies, Electronics and Applied Mathematics, Ecole Polytechnique de Louvain

Dr. Poorvi Vora Associate Professor, Department of Computer Science, The George Washington University

Dr. David Wagner Professor, EECS Computer Science Division, University of California Berkeley

Dr. Dan Wallach Professor, Department of Computer Science, Rice University

Usability Team

- Keith Instone, User Experience Consultant
- Morgan Miller, Usability Analyst, Experience Lab
- Dr. Judith Murray, Research Consultant

Election Auditing

Dr. Philip Stark Professor and Chair of Statistics, University of California Berkeley

Testing Team

Dr. Duncan Buell Professor of Computer Science and Engineering, University of South Carolina

Andrew Regenscheid Mathematician, National Institute of Standards and Technology

Advisory Council

Dr. Ben Adida

Dr. Michael Clarkson Assistant Professor of Computer Science, The George Washington University

Dr. J. Alex Halderman Assistant Professor of Computer Science and Engineering, University of Michigan

Candice Hoke Professor of Law, Cleveland State University

Dr. Ron Rivest Vannevar Bush Professor of Computer Science, Massachusetts Institute of Technology

Noel Runyan Primary Consultant, Personal Data Systems

Dr. Peter Ryan Professor in Applied Security, University of Luxembourg

Dr. Barbara Simons Research Staff Member, IBM Research (retired)

Dr. Vanessa Teague Research Fellow, Department of Computing and Information Systems, University of Melbourne

John Wack Voting Systems Standards, National Institute of Standards and Technology

Dr. Filip Zagorski Assistant Professor of Computer Science, Wroclaw University of Technology

Local Election Officials

Lori Augina Director of Elections, Washington State, Secretary of State

Rachel Bohman Former Hennepin County Elections Manager (Minnesota)

Judd Choate Director of Elections, Colorado, Secretary of State

Dana Debeauvoir Travis County Clerk (Texas)

Mark Earley Voting Systems Manager, Leon County (Florida)

Dean Logan Los Angeles Registrar-Recorder/County Clerk (California)

Stuart Holmes Election Information Systems Supervisor, Office of the Secretary of State (Washington)

Dr. Lois H. Neuman Chair, Board of Supervisors of Elections, City of Rockville (Maryland)

Roman Montoya Deputy County Clerk, Bernalillo County (New Mexico)

Tammy Patrick Senior Advisor to the Democracy Project, Bipartisan Policy Center and Former Federal Compliance Officer Maricopa County (Arizona)

Overseas Vote Foundation Support Team

Susan Dzieduszycka-Suinat President and CEO

Paul McGuire Legal Counsel and Secretary of the Board

Richard Vogt Treasurer and Chief Financial Officer

Capstone Project Team, Carnegie Mellon University, Heinz College, School of Information Systems & Management; Master of Information Systems Management and Master of Science in Information Security Policy and Management: in early 2014, a Capstone Team was assigned to the project team to assist on the Comparative Analysis of E2E systems.

2.3.2 Stakeholder Groups

Although not on the official project team, it was widely acknowledged that there were several communities relevant to the E2E VIV Project outside of those represented on the expert teams and that interaction with members of these communities was essential. These include:

- **Election verification advocates.** Election verification advocates are plentiful, well-informed, and strongly connected. They care deeply about election integrity and verifiability, and unsurprisingly internet voting is a hot-button issue for many of them.

  This negative attitude is compounded by the fact that several vendors have developed internet voting products which are proprietary, closed-source, have never seen a public audit, and are unverifiable. Moreover, many of these vendors make specious claims about the security of their products—claims which the advocate community rejects entirely. Finally, many vendors advocate outsourcing elections entirely to them—a condition that will never be deemed acceptable to the advocate community, even for an end-to-end verifiable internet voting system.

  A small number of advocates are for verifiable internet voting, a small number are adamantly against internet voting of any kind, but the bulk of advocates are on-the-fence. That majority recognize that there are significant scientific and engineering challenges in designing and developing a internet voting system. Moreover, they recognize that the decision to deploy such a system is very much a subjective, political one. For example, in some contexts, it is viewed as perfectly acceptable to use a non-verifiable, outsourced election apparatus (such as Everyone Counts' product); say for the voting of the winner of a reality show. But for government elections of any value, such an option is unacceptable to virtually every advocate.

  Consequently, being fully transparent with—and listening to the feedback from—the election verification advocate community is absolutely mandatory. If the bulk of that community is not swayed by the evidence presented in this report, pursuing any next phase in this project will be fraught with turmoil and will be an uphill battle against a number of influential actors, all with good intentions.

- **Standards Bodies.** Perhaps surprisingly, there is little national or international standardization in the area of elections. A nascent effort to begin standardizing data interchange formats began a decade or so ago and eventually fizzled after only producing one small standard.

  There are a myriad of reasons why this first effort failed. Vendors lobby against, and are disinterested in, interoperability. The EAC's Voluntary Voting System Guidelines (VVSG) are not geared toward a component-based approach to system design, thus there is little cause for defining interfaces and data file formats, since devices cannot be plugged together. Finally, there was unsufficient buy-in from the election research community.

  In 2015 though, this situation changed with the rebirth of the IEEE 1622 committee focusing on elections. The IEEE Voting System Standards Committee 1622 (VSSC/1622) is creating standards and guidelines around a common data format for election data. The aim is that that future election equipment used in U.S. elections and abroad can interoperate more easily. It is the intention of the VSSC that standards and guidelines being developed will be required in future versions of the EAC's VVSG.

Many of the top researchers, election advocates, and election officials in the world are a part of this committee. Additionally, representatives from the major election systems vendors are either participating, or listening in, because they recognize that interoperability will be mandated by future versions of the VVSG.[3]

- How do standards and E2E VIV relate?

- **Vendors.**

- **Hackers and Hacktivists.**

- **Election Officials.**

- **Citizens**

2.4   Methodology

> 1: Joe: Do you want to say anything about these groups? Should we remove this last paragraph and this list? - Susan

> 2: Galois to write this section.

- absorb all input from experts

- read all literature on internet voting

- write requirements and solicit feedback from technical experts

- write personas as foundation for UX studies

- interview LEOs based upon requirements and personas; include in interview information about their current elections framing

- outline report and solicit test and input from experts

- reflect upon latest advances in crypto for E2E VIV

- reflect upon latest advances for reasoning about crypto algorithms, protocols, and implementations

- craft a (set of) architectures and designs that reflect underlying requirements and crypto protocols

- integrate all expert text and input and craft a final report TOC

- solicit more text and reflections from experts based upon final report TOC

- write chapters that were as-of-yet unwritten by experts

- solicit input from all experts on part 1

- solicit input from technical experts on part 2

- gather all input from experts (good and bad, nuances, disagreements, etc.) and capture all in appendices, citations, and footnotes

- polish and release

2.5   Outcome

The E2E VIV Project produced a System Specification Development and Documentation (referred to as a technical report for short) including a Whole Product Solution Specification for an E2E VIV Election System (referred to as a election system for short). The assessment of the system by the expert team has had two possible outcomes.

---

[3]Recall that all election systems in the U.S.A. must be certified at the State or Federal level according to the EAC's voting system testing and certification standards, standards which mandate compliance with the VVSG.

1. Positively, the majority of the expert team may decide that the specified election system meets all of the requirements set forth by the charter of the group. This outcome would indicate that OVF might potentially move forward to ensure that the election system is developed and, potentially, deployed.

2. Negatively, the majority of the expert team may decide that the specified election system does not meet all of the requirements set forth by the charter of the group. This outcome indicates that further funding to design or construct such an election system is, for the moment, unwise and that the community believes that designing a usable and secure election system is still an open scientific, not engineering, challenge.

Fulfilling the usability and security requirements would not be sufficient for a positive assessment by the expert team. A full system specification that is usable and secure may be, for example, far to expensive to build, too difficult to deploy and manage, or mandate too much expertise from election officials to operate. Social non-functional requirements may trump technical functional requirements.

The Whole Product Solution Specification is written in one or more specification languages that cover the technical needs of the E2E VIV Project, particularly with regards to third party high-assurance verification and validation of implementations. Galois recommended using Alloy [10], RAISE [9], or PVS [18] to codify a formal domain model, BON [21] to specify the election system's informal domain model, requirements, architecture, and design, and F [8] and Cryptol [4] to specify election system protocols.

2.5.1 Deliverables Produced

A set of reports and a set of demonstrators were produced. Some elements and report section where non-technical and others not.

Galois contends that all project results should be SMART:

- Specific: the determination of whether a result is accomplished is as objective as possible;

- Measurable: major results have a tracking dashboard on the project website and are updated and reviewed weekly;

- Attainable: E2E VIV Project participants believe they can achieve the results they propose;

- Relevant: results contribute to the priorities, goals, or on-going; operation of the project, and offer clear value to the project; and

- Trackable: progress toward the achievement of a result is monitored, including project budget.

Moreover, each result must have a customer. A customer is an individual or group who will negotiate a result and who will be actively engaged. They will truly care that the result is achieved and work to make us all successful in doing so. The customers of the E2E VIV Project are the Local and State Election Officials.

2.5.2 User Interface Design

The user interface (or UI for short) of the E2E VIV election system is the critical factor in ensuring that the system is simultaneously usable, accessible, and secure. Consequently, a detailed UI design informed by usability (UX, for short) and accessibility testing is a mandatory component of a detailed system specification.

Usability and accessiblity testing is a key component of this report, and the outcome of that testing is meant to inform the UI design of any future E2E VIV systems. As such, most of the effort relating to UI and UX within the project is focused on developing a technical infrastructure and process for the efficient definition and execution of qualitative and quantitative usability and accessibility studies.

In order to effect these goals, a demonstration system that mimics a voter's interaction with an E2E VIV system was developed by Galois. That system is a variant of the STAR-Vote system designed by Wallach et al. []. STAR stands for Secure, Transparent, Auditable, and Reliable. STAR-Vote is an end-to-end verifiable ballot marking device. As such, it is not designed for, or meant to be used for, internet voting. But insofar as its voting process is identical to that of most of E2E VIV election schemes in the literature, it was decided to use it as a demonstration vehicle for usability and accessibility experiments.

The Galois STAR-Vote implementation has a web-based UI, thus can be used and demonstrated remotely for interactive and non-interactive experiments to gather both qualitative and quantitative feedback. Several variants of STAR-Vote have been implemented for UX testing. These variants include simple changes—like different typeface choices and sizes, background colors, supporting images, help text, mouse pointer graphics, etc.—as well as more complex changes—like different voter, challenge, and audit workflows.

In an interactive, qualitative experiment, a facilitator and a voter communicate using a video chat system such as Skype and the voter shares their desktop with the facilitator. Optimally, the facilitator is someone who is deeply familiar with the issues of E2E VIV systems, is familiar with STAR-Vote, and has expertise in usability and accessibility. The voter then uses (one of several variants of) STAR-Vote, voicing their thoughts and feelings about their experience in real-time. After the voter has completed their participation in the demonstration election, the facilitator uses a script to query them about their impressions.

For a non-interactive, quantitative experiment, voters will be solicited via social media, mailing lists, etc. to experiment with (variants of) STAR-Vote. Sample voters in these experiments are given ample information about what kinds of information is being collected about their behavior so that they can make a fully-informed judgement about their participation.

Various quantitative measures related to voter participation and interaction are measured automatically, both within their web browsers and on the STAR-Vote server. Must of this data is akin to the analytics that any professional website collects about its users: How do voters navigate the site? Where does a voter pause for a long time and read? When does a voter ask for help? When does a voter hover over a button a long time before they decide to click it? How often do voters challenge ballots or verify their votes? How often do voters examine the bulletin board? Is there a correlation between the interactive behavior of a voter while voting and their likelihood of voting, challenging, or auditing correctly?

=wrap up=

2.6   Next Steps

15

Chapter 3

Remote Voting (Philip) (100%)

3.1 Rationale

Remote voting is becoming increasingly common, prompted by the growing and diverse needs of voters. Remote voting is used to enable overseas citizens and military personnel to participate in elections, reduce access related discrimination domestically, and decrease expensive administrative overhead of polling locations. In the United States, fewer than 5% of ballots cast in general elections during the 1980s were cast before election day. By the 2012 general election, 31% of all ballots were cast early, and 17% were cast by mail. The states of Washington, Oregon, and most recently Colorado have entirely switched over to all-mail voting. For an election system fully enfranchise the electorate, it must treat remote voting as a first-class capability rather than as a backup system with second-class effectiveness, speed, security, and integrity.

3.1.1 Accessibility

According to the International Center for Disability Information and the National Institute on Disability and Rehabilitation Research, 20% of Americans live with disabilities. The Voting Accessibility for the Elderly and Handicapped Act of 1984 mandates that any person with a disability may vote remotely without having to present medical documentation, reducing the barriers to remote voting for those with accessibility needs.

3.1.2 UOCAVA

In 1986, Congress enacted the Uniformed and Overseas Citizens Absentee Voting Act (UOCAVA) to address the needs of citizens in the uniformed services, merchant marines, and other overseas civilians. UOCAVA mandates that these voters be able to register and vote remotely in federal elections. It is difficult to calculate the exact number of UOCAVA eligible voters, but Figure 3.1 lists a recent estimate of the total.

3.1.3 Domestic Absentee

Domestic absentee voters are those who vote early in-person, or cast their votes cast by mail because they are unable or do not want to be present at polling locations on election day (this excludes UOCAVA voters, and voters in states that vote exclusively by mail). 21,853,762 or 16.6% of all votes cast in the 2012 US general elections were from domestic absentee voters [1]. As of 2015, 27 states allow voters to apply for an absentee ballot without providing a justification, known as "no-excuse absentee voting".

5: Ask Philip for context on these numbers. Statewise breakdown doesn't seem to add much, and the difference between what the two are counting is unclear

State	Overseas Voting Eligible Population (McDonald 2009)	Overseas military and federal civilian employees (US Census Bureau 2010)
Texas	11.05%	11.78%
California	9.78%	8.44%
Florida	9.09%	9.54%
New York	5.31%	4.12%
Pennsylvania	4.10%	3.12%
Illinois	4.03%	3.24%
Ohio	3.51%	3.07%
Michigan	3.29%	2.68%
Georgia	2.84%	3.83%
Washington	2.78%	2.77%
North Carolina	2.78%	2.91%
Tennessee	2.57%	2.81%
Virginia	2.51%	3.52%
Estimated Total	4,972,217	1,042,523

Figure 3.1: Comparisons of American Overseas Population by State

6: Take out this figure too?

3.1.4 Expectations

In 1952, a study by the American Political Science Association defined ten voting rights necessary for members of the armed services [6]. Although initially defined for military voters, these rights have served as the basis for defining the expectations of all remote voters through UOCAVA and other subsequent legislation. Among these rights are:

1. To vote without registering in person.

2. To vote without paying a poll tax or having to meet unreasonable requirements

3. To use the Federal postcard application both to register and to request a ballot, rather than having to use state-specific paperwork.

4. To receive ballots for primary and general elections in time to vote.

5. To be protected in the free exercise of their voting rights.

6. To receive essential information needed to vote.

These rights first found their way into law in the Federal Voting Assistance Act of 1955 (FVAA), but due to partisan struggles, the rights were watered down from requirements into recommendations, leaving much of the decisions about final implementation to the states. Over time, subsequent legislation has strengthened these recommendations into guarantees and requirements, and has expanded rights to include participation by non-English speakers and people with disabilities.

State	Percent of Population
Colorado	71.4%
Arizona	65.9%
Montana	57.5%
Georgia	48.8%
Iowa	43.1%
California	39.8%
Hawaii	36%
North Dakota	28.8%
Florida	26.8%
Michigan	26.4%
Wyoming	26.2%
Maine	25.5%
Nebraska	25.4%
Idaho	24.3%
Ohio	22.4%
Wisconsin	21.4%
Vermont	20.4%

Figure 3.2: Votes Cast as Domestic Absentee 2012 General Election

3.2 History

3.2.1 Armed Forces Voting

Before the American Civil War, US citizens primarily voted in their places of residence, and many states legally barred the casting of votes from outside state borders. There was little effort from any state to accommodate absentee voting. However, in 1864, with the Civil War displacing soldiers from their residences, Lincoln's re-election was at risk. With much lobbying on behalf of the Republican Party (and opposition from the Democratic Party), nineteen Union states adopted absentee voting procedures for military voters in time for the election. Since the motivation for passing these laws was to secure Lincoln's re-election rather than permanently expand voting access, many absentee military voter laws were treated as temporary and repealed after the war.

For the 1918 midterm elections, the US War Department decided that it was not ready to support the military vote, going so far as to prohibit individual states from canvassing overseas soldiers serving in the First World War. The Second World War inspired another push for the military vote in hopes of supporting the re-election of the presidential incumbent. This prompted the Soldier Voting Act (1942) which, although passed too late for the 1942 midterm elections, gave military personnel absentee voting rights for federal elections during times of war without subjugation to voting tax or postage costs. Nonetheless, the act was notable for mandating that all overseas voting would be regulated at the federal level and implemented at the state level, a structure that continues to this day. By 1944, partisan politics led to the weakening of the state mandate from a requirement to a recommendation, leading to a 29.1% turnout rate vs. the 60% domestic turnout rate [51].

3.2.2 Remote Civilian Voting

Progress for civilian absentee voters lagged behind progress for military voters. In 1896, states began introducing civilian absentee voting legislation. By 1924 only three states had no absentee voting legislation, but state laws were a confusing and inconsistent patchwork, lowering absentee turnout. Major progress for civilian absentee voters would only come with legislation motivated primarily by military voters such as the FVAA.

In the 1960s, lobbying from overseas civilian groups led to amendments to the Voting Assistance Act expanding the number of civilians covered by the law, though those amendments were once again voluntary recommendations to the states. As lobbying pressure increased further, the Overseas Citizens Voting Rights Act (OCVRA) passed in 1974 for the first time guaranteed, rather than merely recommended, absentee voting rights for overseas civilians.

In 1986, the Uniformed and Overseas Citizens Absentee Voting Act (UOCAVA) was passed, combining and replacing the FVAA and the OCVRA and making the rights recommended by the previous acts into requirements for both military and overseas civilian voters.

3.2.3   Disabled Civilian Voting

The Voting Rights Act (VRA) of 1965 was the first legislation to enfranchise voters with disabilities. The VRA granted voters who require assistance to vote by reason of blindness, disability, or inability to read or write assistance by a person of the voter's choice.

The Voting Accessibility for the Elderly and Handicapped Act of 1984 (VAEHA) was passed to improve access for disabled and elderly individuals. However, as with the recommendations of FVAA and OCVRA, states were left to set their own standards of access, and limited the disabled voters group to those with *physical disabilities*. The VAEHA did, however, mandate "no notarization of medical certification shall be required of a voter with a disability with respect to an absentee ballot or application for such ballot."

The 1990 American Disabilities Act (ADA) requires that people with disabilities have access to basic public services, including the right to vote. The ADA does not strictly require that polling locations are accessible. however it did extend the definition of disability to:

> "a person who has a physical or mental impairment that substantially limits one or more major life activities, a person who has a history or record of such an impairment, or a person who is perceived by others as having such an impairment."

The majority of federal laws passed to protect voting rights of disabled citizens have struggled to clearly define a representative range of disabilities, and are often focused on in-person access to physical polling locations, which is expensive for states to implement. Additionally, state-defined policies often ignore rights to voting privacy, and exclude persons with multiple disabilities for which assistive technologies are not yet available.

3.2.4   Modern Remote Voting

In 2002, Congress passed the Help America Vote Act (HAVA) in response to problems found in gathering, counting, and auditing votes in the 2000 presidential election. HAVA requires that all polling places in elections for federal office anywhere in the United States have at least one voting system capable of assisting disabled voters, addressing some accessibility concerns.

HAVA is also a response to the large number rejected ballots in the 2000 election and an inability to sufficiently audit ballots. HAVA recommends election systems produce a Verifiable Voter Paper Audit Trail (VVPAT) while preserving the privacy of the voter and the secrecy of the cast ballot. HAVA also created the Federal Election Assistance Commission (EAC) to oversee the development of new voting machine standards, and released the Voluntary Voting System Guidelines to aid in this transition.

The Military and Overseas Voter Empowerment (MOVE) Act of 2009 addresses barriers to overseas voter participation, specifically attempting to reduce the number of ballots that were not counted due to arriving late. The MOVE Act requires states to transmit absentee ballots at least 45 days before election day, make all registration material and blank ballots available electronically, and remove notarisation requirements on all election material.

Historically, legislative progress on absentee voting moves slowly. Existing voter rights regulations are enforced at a state level and are hampered by local political attitudes. In 2010, the Uniform Law Commission, a non-profit organization, oversaw drafting of the Uniform Military Services and Overseas Civilian Absentee Voters Act (UMOVA). UMOVA is designed to identify and standardize the important protections and benefits found in federal legislation like UOCAVA and MOVE in state and local elections. As of April 2015, fourteen states and the District of Columbia have enacted UMOVA.

3.3 Shortcomings of Current Practice

Despite years of progressively stronger legislation addressing the needs of remote voters, there are many shortcomings of current election practices. The topics listed below draw from specific concerns that have a large impact on remote voting participants.

3.3.1 Use of Communication Technologies

The majority of remote voting takes place via postal mail which is subject to many inherent faults that are exacerbated for voters. The 2008 Post-Election UOCAVA Survey Report and Analysis found that 52% of attempted UOCAVA votes were not counted due to problems in the mail delivery process. Additionally, maintaining correct voter registration information for military voters and others who frequently move to new addresses while abroad is expensive and prone to error.

To address differences between states' absentee registration and voting practices, the Federal Voting Assistance Program provides an Voting Assistance Guide for registration forms. Unfortunately, this is very large and difficult to follow. For states without streamlined online registration, this has resulted in many failed voter registration attempts.

7: doesn't the Federal Postcard Application address this?

3.3.2 Accessibility and Usability

In 2007, 20% of Americans with disabilities said they were unable to vote in presidential or congressional election due to barriers at or getting to the polls [48]. This is frequently a consequence of the voting technologies used and the physical location of polling places. In the 2000 presidential election, 56% of randomly sampled polling places in the United States had at least one identified impediments for disabled voters [42].

For in-person voting, disabled persons often forfeit privacy in order to receive assistance from an aide, often because polling locations have of insufficient assistive technology, or because the assistive technology in place is too difficult for both voters and poll workers to use. Remote voting still presents obstacles: those with dexterity impairments often have problems with handling and marking paper absentee ballots.

3.3.3 Auditing

Although it is believed that voter fraud is fairly uncommon, it is a major concern in a bipartisan system. It is very difficult to detect voter fraud without depressing turnout or disenfranchising legitimate voters. Policies intended to reduce fraud or protect identities, such as increasingly prevalent and strict voter ID requirements, often result in a higher rate of rejected ballots even for remote voters. In the 2012 general election, over 20% of absentee ballots were rejected due to non-matching signatures or unmet ID requirements [1].

3.3.4   Voter Privacy

Voter privacy is a key expectation of a fair voting system. Privacy promotes voter independence, and helps prevent voter coercion and vote buying. Most remote voting practices require that voters forfeit independence or privacy due to the physical reality that election officials can enforce privacy in polling places, but not in voters' living rooms. Several states even require signing a voter privacy waiver when casting a remote ballot [52].

Chapter 4

E2E VIV Explained (Philip/Daniel/Adam) (95%)

4.1 Goals

Typical Internet voting election processes have six phases:

Setup During the setup phase, the election officials gather the information needed to run an election. This includes gathering registration information for all voters, identifying the issues and races that will be voted on, designing and specializing ballots, sending instructions and other information about the election to voters, and so on.

Distribution Once the election has been set up, election officials must distribute ballots to the voters. Different voting system architectures use different mechanisms, including postal mail, email, or by having voters interact with a website.[1]

Voting Voters then fill out their ballots, often with the help of software installed on their own computers.

Casting Filled out ballots are then returned to the election officials; as with distribution, different architectures use different mechanisms.

Tallying The tallying phase includes the remainder of the election finalization tasks: counting votes and announcing the election outcome are common to almost every process, though some include other miscellaneous tasks like publishing certain information needed for audits.

Auditing Some elections will inevitably be disputed; in such cases, there is a final phase in which interested parties look for evidence that the election outcome is correct (or not!).

One major concern for Internet voting involves ballot integrity during the distribution, voting, and casting phases. For the election outcome to be correct, it is important that the ballot that is received by and displayed to the voter match the ballot that was created and sent by the election officials; that the computer used to fill out the ballot faithfully reports the intention of the voter; and that the filled out ballot be received by the election officials exactly as it was when it was sent by the voter. Typical Internet communications involve not just the computers owned by the two parties communicating, but also many intermediary computers controlled by neither party. A good election system needs to account for this, making it impossible for these intermediaries to intercept ballots for viewing or modification during transit. Another concern is that voters computers are rarely administered by experts, and as a result many of them

8: signposting; some content is here, but we need an intro and transitions explaining what content is about to happen

---

[1]We distinguish between sending instructions to voters and distributing ballots; there is no hard and fast rule for the distinction, but a rule of thumb is that instructions are applicable to many voters, whereas anything that has been specialized for a single voter is part of the ballot and falls under the distribution phase.

are compromised by outside forces. One consequence of this is that the voting phase itself may become corrupted: even if the ballot arrives unchanged at the voter, malware on the voter's computer may change the way the ballot is displayed or the way the vote is recorded before casting the vote. It can be difficult to design a system that is resistant to this kind of attack without seriously sacrificing the usability of the system. Some systems use alternative distribution mechanisms as cross-checks; for example, sending something to the voter by postal mail which can be used to check that the ballot displayed by their computer is correct.

To the extent that it is possible, it is desirable for Internet voting to be private and anonymous. Voters should feel comfortable voting the way they like (and not feeling pressured to vote for a particular candidate or to vote a particular way on some issue); the fewer people who know or can find out the way a given voter voted, the more comfortable they can feel. On the other hand, election officials only want to record votes from people who are registered to vote, and even then want to record only one vote from each voter. Thus there is a tension during the vote casting phase between retaining the anonymity of votes and ensuring that a vote is coming from somebody who ought to be able to vote.

One popular approach to this problem in existing systems is to initially require each vote to be tied to the voter who cast it long enough to decide whether to include the vote in the later tally or not; then to keep the vote but delete the information about who cast the vote. This approach can work; however, audits of systems that take this approach shows that it is all too easy to accidentally retain the connection between votes and voters longer than intended, and make this information much more widely visible than intended. From the privacy side of the tradeoff, it would be better if the voter could be confident that there was no connection stored because the information they send to the election officials during the casting phase does not include any personally identifying material.

There is a subtle distinction being made here. We certainly want our Internet voting systems to be correct, private, secure, and so forth. It is important for the people developing these systems to verify that they are correct and take an active role in seeking out and eliminating defects in the system. But the goal of verifiable Internet voting is to go even farther: not just correct, but *visibly* correct. That is, it must be possible for the parties using the system to be able to *check* that the system is behaving correctly, without trusting in the abilities of the people who created the system to avoid bugs or trusting in the inability of third parties to influence the behavior of the system. As applied to anonymity: since it is not easy to prove to somebody else that you have deleted some information, one must simply avoid giving them that information in the first place. This theme—of being not just correct, but verifiable—is one of the central ones of verifiable Internet voting, and is a critical part of the defense against the software bugs, security vulnerabilities, and sophisticated cybercrimes that history tells us are sure to crop up.

The tallying process provides a particularly good example of the difference between correctness and verifiability. We certainly want the election system to count the votes correctly; but the goal of verifiability is to provide some evidence to voters that the election outcome is correct. For example, some systems allow voters to check that their vote was included in the election outcome; some allow voters to check that the system is recording the content of their votes correctly; some even allow voters to check that the number of people that voted for a given candidate is accurately calculated without revealing any of the individual votes. Meeting these verification goals without violating the anonymity and privacy goals can be a balancing act.

Each of these individual goals contribute to a single top-level goal: end-to-end verifiability. The "end-to-end" property is that the whole election process produces a result that matches the intentions of the voters. The subgoals of this are summarized with the catchphrase, "Cast as intended; recorded as cast; and counted as recorded." This recapitulates the concerns discussed above; "cast as intended" is the demand that casting use secure communications and other mechanisms to ensure that malware and outsiders cannot change the vote, "recorded as cast" is the demand that the election system itself correctly interprets a vote, and "counted as recorded" is the demand that the tallying process be faithful; and all of these demands are subject to not just correctness but verifiability, so that a voter can convince themselves that these properties hold even if they suspect that the system or election officials have been corrupted.

4.2   Shortcomings and Expectations of E2EVIV

As discussed in Chapter 3, there are several difficulties with current voting processes: voters with disabilities cannot vote unassisted, communication channels with remote voters are slow and unreliable, vote tallying is labor-intensive and error-prone, and election audits are costly. Additionally, there is little visibility into the election process, meaning that individual voters and, in some cases, even auditors, must trust the reports of election officials and voting hardware vendors on election outcomes and processes. Internet voting may be able to alleviate some of these concerns. Voters with disabilities could potentially use their own familiar hardware, such as Braille displays, screen readers, sip-and-puff input devices, and so on, to participate in the election. Internet communications are traditionally speedy (seconds per message rather than weeks) and relatively robust compared to overseas postal mail. In most systems, tallying is automated and fast. Auditing can still be a challenge, though there is some hope that verifiable systems can make elections more transparent for this purpose, too.

There are also some serious challenges in rolling out an Internet voting system. Chapter 7 discusses the feasibility of producing a system that meets the security and verifiability goals we have touched on above. In addition to those concerns, the ability of normal voters to use the system to cast their vote in the way they intend to vote is a major goal; as we discuss below, current systems do not meet this goal very well. One component of this is the system itself; though another that is common to all Internet voting systems is the need for voters to have Internet access. This is not necessarily possible for all overseas and military voters.

> 11: would be nice to have some hard data about this

4.3   E2E VIV in Practice

A number of practical voting systems have been developed based on the principles of E2E VIV. This section describes several systems that have been used in a real election or in a pilot.

4.3.1   RIES

RIES, the Rijnland Internet Election System [33], was first used in 2004 to support elections to the Rijnland water management board, supplementing the system of postal voting used by the water board. A subsequent version was used to allow expatriate voters to participate in the Dutch parliamentary elections [31].

Before a RIES election, credentials are mailed to every voter in the form of a very long number. The same mailing also includes instructions for the voter.

During the election, voters log into an election web site that includes a client-side voting application written in JavaScript. The client-side application encrypts the vote by passing the voter authorization code and the public ID of the candidate through a one-way function to create the encrypted vote. The encrypted vote is then placed on a public bulletin board that serves as a ballot box.

At the close of the polls, the election authority releases the final vote tallies along with a codebook containing the encryptions of all valid credentials with all candidate IDs.

The algorithms and protocols used RIES are public, and each voter, having access to all of the inputs and outputs, may (in principle) check the computations. This is weaker than the desired individual verifiability, but nonetheless, far stronger than conventional voting systems.

The Organization for Security and Co-operation in Europe (OSCE) sent an election assessment team to observe the use of RIES in 2006. Their report contains observations of critical security features of the system that could not be observed [43]. Further weaknesses were revealed by the Eindhoven Institute for the Protection of Systems and Information (EiPSI) in 2008 [34], notably that:

- the procedure of voter self-check is quite complicated,
- the two-channel (mail and Internet) voting makes system less transparent,

- too much power is given to the election administrator and the system's Internet host,

- issues arise when modifying the codebook due to a revoked ballot, and

- there are realistic ways to forge votes via cryptographic hash collisions.

One of the more important lessons learned through RIES is that when voter authorizations are distributed long in advance of the election, a mechanism must be provided allowing voters to obtain replacement credentials and invalidate lost credentials. These mechanisms add significant complexity to system, and is a source of some of the problems reported in the OSCE and EiPSI reports.

Another feature of RIES rife with tradeoffs is the ability to perform testing during the election: pre-invalidated test ballots are deliberately added to the bulletin board in order to test the network path from selected Internet clients to the server. While such testing in principle can increase confidence in the election integrity, in practice it opens the system to spoofing and denial of service attacks. Furthermore in the RIES implementation the system is aware of the fact that it is processing a testing ballot, and all of the test ballots were voted identically from the same computer, limiting the confidence added at the expense of these vulnerabilities.

In the wake of these critical reports, plans to use RIES in the 2008 Dutch parliamentary elections were scrapped, and Internet voting as a whole was banned in the Netherlands.

4.3.2   Prêt à Voter

The state of Victoria in Australia held a governmental election in November 2014, using a version of the Prêt à Voter system [11, 17]. An attempt was also made to use Prêt à Voter in a student election at the University of Surrey in February 2007 [8]. The failure of this attempt illustrates many of the pitfalls of adapting a research system to an actual election, such as a short timetable, a lack of clear requirements, and the need for rigorous implementation practices.

Prêt à Voter uses two-part paper ballots with the candidate names on one part and the voting targets plus a ballot ID number or barcode on the other part. Typically, the two parts are printed as a single sheet with a perforation to divide the sheet after voting.

From the voter's perspective, the order of the candidate names on the ballot appears to be random. The voter marks her choice next to the candidate name of her choice, separates the two parts of the ballot, and destroys the candidate names. She may take a copy of the voted part home for later verification.

For tabulation, there is a cryptographically secure mapping from the ballot ID numbers to the apparent random order of the candidate voting positions. Multiple custodians using a mixnet or similar technique use this mapping to decode cast ballots into anonymized plain-text ballots which are then posted to a bulletin board.

Unvoted ballots may be audited before, during and after the election to ensure that the decoding of cast ballots is being correctly performed. Randomly selected stages in the decoding can be challenged to prove the integrity of the count, and the plain-text decoded ballots are easily counted for verification by any interested party.

An individual voter may also search for their voted ballot ID on the bulletin board. This reveals the positions that were marked on that ballot, but crucially, it does not show the corresponding candidate names. The voter may therefore verify that the positions marked at the polling place were correctly recorded by the election officials, but because the voter no longer has the part of the ballot linking candidate names to ballot positions, the voter cannot prove to anyone else how the ballot was voted.

4.3.3   Punchscan

Punchscan [46, 47] was used for the graduate student association elections of the University of Ottawa in 2007 [25]. It is likely the first E2E voting system with ballot privacy used in a binding election.

The election experience for a Punchscan voter is very similar to that of Prêt à Voter. The system uses a two-part paper ballot where the top part has candidate names and candidate numbers (or letters) and the bottom part has numbered (or lettered) voting targets. Holes punched in the top part expose the voting targets below. The order of the voting targets for each race appears random to the voter. Both halves of the ballot bear an identical serial number.

The voter casts their vote by marking her choice with a bingo dauber, and the two halves are separated. Either side can be scanned (since the bingo dauber marked both through the hole and around it) as the cast ballot. The other side is destroyed, and a copy of the cast side may be retained by the voter.

A curious voter may inspect the public record of any cast ballot exactly as with Prêt à Voter. It does not matter which half of the ballot the voter retained, because there is no public display of the numbers that link candidate names to voting positions; only the position that was marked is displayed. Again, individual ballots may be audited, and the key to tabulating the votes is that there is a cryptographically secure mapping from the ballot serial numbers to the apparent random order of the candidate voting positions.

4.3.4 Scantegrity II

Scantegrity II (Invisible Ink) [19, 20] was used in the Takoma Park, Maryland municipal elections in 2009 [13]. In 2011, it was used for in-person voting with Remotegrity (Section 4.3.5) used for absentee voting. The 2009 Takoma Park election was the first use of an E2E system with ballot privacy in binding governmental elections.

Before the election, officials generate the seed to a pseudorandom number generator using a secret sharing scheme. Three-letter alphanumeric codes are created for each choice on each printed ballot using this seed, and additional tables are created so that interested parties can later confirm that the tally was computed correctly.

During the election, the voter experience is nearly identical to that of conventional optical-scan paper ballots. When the voter marks their choice, the ink in the pen reacts with invisible ink on the paper to disclose the three-letter code in the marked voting target. The ballot ID number and the displayed code are posted to a public bulletin board.

After the election, public verification of the final tally proceeds with the public bulletin board in a manner similar to that of Punchscan and Prêt à Voter.

In addition to the public verification, an individual voter who takes note of their ballot ID number and the code revealed from invisible ink may use the public bulletin board to check that their ballot was indeed tabulated, though this information is not sufficient to prove that they voted a particular way.

4.3.5 Remotegrity

Remotegrity [64] is a remote coded voting system that was used for absentee voting alongside Scantegrity (Section 4.3.4) for in-person voting for the 2011 Takoma Park, Maryland municipal elections.

Remotegrity voters receive a coded voting ballot and an authentication card in the mail. The codes on the ballot can be covered by a lottery-style scratch-off field. The authentication card contains several authentication codes under scratch-off, a lock-in code under scratch-off, and an acknowledgment code. Both cards have serial numbers. The voter can be sent two ballots so that she can use one for auditing purposes.

To vote, the voter enters both serial numbers, the codes corresponding to her choices, and an authentication code obtained after scratching-off a surface chosen at random.

She returns to the election website a few hours later to check if her codes are correctly represented, and to see if the election authority has posted her acknowledgment code next to the codes. This indicates to her that the election officials received valid codes for her ballot.

She scratches off the lock-in code and posts it on the website. This affirms to the election officials, observers and other voters that her vote is correctly represented on the website.

Among all of the systems discussed here, this is the first one that asks the voter to take positive action to confirm that the vote was correctly posted.

As with RIES, if we assume that there is no communication between the computer used to print the credentials and the computer used to collect the votes, the latter computer does not know the mapping from codes to candidates, so the vote is not revealed to the computer. Further, because the computer does not know a valid code corresponding to another candidate on the ballot, it cannot change the vote. Finally, and uniquely, because the computer does not know the acknowledgment code, its presence on the election website assures the voter that the election officials received a valid code for her ballot.

The tally is computed from the codes in a verifiable manner that corresponds to the coded voting system used.

If a jurisdiction is nervous about using the Internet for remote voting, Remotegrity ballots can be mailed in, and voters can check for their codes on the election website to be assured that their vote correctly reached election officials.

4.3.6   Helios

Helios [3, 4] is a system developed for web-based Internet voting. It was used for the election of a Belgian university president in March 2009 and by numerous universities and associations since then, including the Association for Computing Machinery and the International Association for Cryptologic Research.

Before a Helios election, officials input the email addresses of the voters who will be participating. The system emails the voters their randomly-generated login information and the link to the election website.

During the election, the voter enters their choices on the website. After entering her choices, the voter has an option to spoil their ballot in order to verify that it was recorded correctly. Upon completing a non-spoiled ballot, the system sends an email confirming the receipt of their vote, though not their choices. At any time before the close of the election, the voter can repeat these steps and the new vote will replace the old vote.

After the election, Helios uses homomorphic vote tallying with the optional addition of mixers and mixnets in some derivatives [10, 57].

Voter authentication is not required until after the voter decides to cast the ballot, so any interested party may prepare and audit ballots. All cast ballots are posted in encrypted form on a public bulletin board so that voters may check that their ballots have been correctly recorded. Similarly, after the polls close, the decryption and vote tally may be checked.

4.3.7   Norwegian System

Between 2011 and 2014, the Norwegian government ran an Internet remote voting trial [29] using a cryptographic protocol designed by Scytl, a commercial voting system vendor. Scytl and the Norwegian government assert that this is an E2E system, which if accurate is the first effort by commercial voting system vendors to enable E2E elections.

The Norwegian system uses a three-channel model involving postal mail, the Internet, and SMS text messaging. Before the election, the voter receives authorization codes to cast a ballot via postal mail.

During the election, the voter uses a computer to cast an encrypted ballot. The voter can cast multiple ballots; only the last ballot cast is counted, and if a voter votes both on paper at a polling place and by Internet, the paper ballot overrides the Internet ballot. After casting a ballot, the voter receives a confirmation code offering a partial end-to-end proof via an SMS message.

Available descriptions of the Norwegian system are incomplete, so it is not possible to analyze the system in depth. However the system's claims to protect voter privacy are weak: "If the voter's computer and the return code generator are both honest, the content of the voter's ballot remains private." In addition, the receipt delivered to the voter proves only that the encrypted ballot was received as cast, not that it was counted as cast or that the encrypted vote matches the voter's intent.

The system evolved significantly between its first use in 2011 and 2013, with added complexity to attempt to assure voters that their ballots were stored as cast. In 2013, the Carter Center mounted a serious effort to observe the Norwegian system in action. Their report on the operation of the system and the problems they had observing it offers useful insight into the administration of E2E systems in general as well as the particulars of the Norwegian system [14].

4.3.8 Wombat

The Wombat voting system [32] has been used for multiple pilot elections in Israel. It is an in-person voting system where the voter votes on a touch-screen and obtains a printout of her vote with an encryption of it. The voter can choose to cast or audit the encrypted vote. If she chooses to audit the vote, she may check if the vote was correctly encrypted. If she chooses to cast it, the ciphertext is posted online, and she casts the unencrypted vote in the ballot box (this may be manually counted) and takes the ciphertext home. The votes are tallied using a verifiable mixnet.

4.3.9 DEMOS

DEMOS [24] is a coded vote system where the voter is given a two-part coded ballot; she audits one part and uses the other to vote. Associated with each choice on the ballot is a vote code—the encryption of the vote, which is entered in the voting machine by the voter, and a receipt code which the voter does not enter, but which is posted online next to the vote code.

The voter can check the receipt to ensure her vote reached the election authorities. The ballot also has a QR code containing all the information on the ballot which can be scanned by the voter if she prefers not to manually enter the vote code. Once the ballot is entirely represented on the computer, the voter can then make her choices. Note that if the voter scans the QR code, the scanning computer knows how she voted. The vote codes represent homomorphic encryptions of the votes and the verifiable tally is obtained in a standard manner.

A pilot study of DEMOS was carried out during the 2014 European Elections in Greece.

4.4 Limitations of Existing Systems

E2E systems inherit many of the limitations of traditional voting systems. Reliability of equipment, reliance on procedure, trust in insiders, and accessibility are all problems with traditional in-person voting systems. For remote systems, the integrity of postal systems, turnaround time for mailed materials, access to Internet or fax technology, and reliability of Internet servers are all well-documented obstacles to voting.

12: reference other sections about this

Existing E2E systems mitigate some of these limitations. For example, code voting limits the ability for attacks against postal mail systems to change the candidates marked on voted ballots. However if an attacker simply intercepts and destroys the voted ballot, a replacement might not arrive in time for that voter to participate in the election. To mitigate this, election officials might choose to instead accept voted ballots via fax, email, or website, but such expedient measures often trade off the verifiability that makes an E2E system desirable in the first place.

In this section, we examine the limitations of E2E systems with a particular focus on the limitations that are unique to or exacerbated by E2E characteristics.

4.4.1 Voter Secrecy

Systems like Prêt à Voter (Section 4.3.2) and Punchscan (Section 4.3.3) rely on a randomized candidate order or a code on printed ballots to ensure voter secrecy. Voted ballots must appear on a public bulletin board in order to verify the election results, and so to protect secrecy only the selected position or code is visible on the final ballot along with a ballot ID.

If an insider is able to review the printed ballots before the election, they can record how the candidate positions are arranged for each ballot ID and therefore identify which candidate is marked on the voted ballots, thus violating secrecy [11].

Recent writing on Prêt à Voter recommends printing ballots on demand at polling places in order to limit this possibility [49]. Printing on demand introduces additional problems and expense compared to centralized printing. More printing equipment is required at each polling place, that equipment can break or be difficult to operate, and the printing equipment must have some way of communicating with the rest of the election infrastructure to ensure it has, for example, the correct cryptographic seeds for generating new ballots.

Scantegrity II (Section 4.3.4) uses invisible ink to hide the vote codes on unvoted ballots, and Remotegrity (Section 4.3.5) can use scratch-off fields to hide vote codes and other information required to cast a ballot. These techniques limit the opportunity for insiders to learn secrecy-compromising information without being detected through the presence of a marked or damaged ballot.

Even with techniques to mitigate insider foreknowledge of the ballots, secrecy still can depend on voters and poll workers correctly following procedures. A voter can leave the polling place with a complete Prêt à Voter ballot, for example, failing to shred the half with the candidate order. With both halves of their ballot, they can prove how they voted, losing receipt-freedom.

RIES (Section 4.3.1) makes a deliberate secrecy tradeoff by weakening the receipt-freeness requirement in exchange for providing universal verifiability and a degree of individual verifiability. The results of an entire election can be independently audited with only the information publicly available after the election. However if a voter discloses her credential or her encrypted vote, the same public information may be used to violate ballot secrecy. The developers of RIES judged this violation to be no more severe than the threats to ballot secrecy inherent in postal voting, and therefore worth accepting for the benefit to verifiability.

4.4.2 Ballot Stuffing

As when ensuring voter secrecy, many E2E systems depend on correct procedures to defend against ballot stuffing. For example, during the University of Ottawa elections using Punchscan, more ballots were cast than voters recorded in the pollbook. In this case, ballot stuffing can be caught after the fact by poll workers, but is not an inherently verifiable property of the system, and requires trust in the accuracy of the poll workers.

In the Helios system, officials can enter voters by email address, and so there is limited protection against insider ballot stuffing. Helios relies on individual voters verifying their votes, with little provision for an interested party to verify the entire election, making it difficult to detect this type of fraud [50].

A pre-election step that publicly publishes tables of valid ballot IDs can help mitigate this problem, but also creates others. All votes in the final tally have an (anonymized) provenance that can be traced back to before the election began and presumably cross-checked with voter registration rolls. However having a fixed set of ballot IDs can make it harder to replace lost, stolen, or spoiled ballots, or to providing for late or same-day voter registration.

4.4.3 Infrastructure & Equipment

Election equipment fails in practice. An E2E system must be resilient to failures while not giving up E2E properties. A system that lacks robust fallback mechanisms is not itself robust, but is only as strong as its weakest fallback. For example, if a remote voting website fails and election officials resort to accepting voted ballots by email, E2E guarantees are lost for all of the emailed ballots.

In addition to being more sensitive to failures, verifiable election systems often require more sophisticated equipment than traditional systems. For in-person voting, a verifiable system might require ballots to be printed on demand, a high-quality shredder for two-part ballots, and more sophisticated assistive devices. This complexity incurs additional cost and poll worker training requirements.

Many E2E systems post encrypted ballots during an election to a public bulletin board. In order to update the bulletin board in real time, these election systems are distributed systems, networked via traditional means or via a manual air gap. Depending on the networking scheme this can open equipment to distributed denial of service (DDoS) attacks, network partitions, inconsistency, and other problems inherent to distributed systems.

Internet systems compound the difficulties of distributed systems by requiring the systems to be accessible via the public Internet, increasing the possibilities for DDoS and other malicious attacks. Furthermore, many systems allow voters to use their own computers to vote, leading to pitfalls inherent when election officials lack control over the voting environment. Malware on the voter's computer might undermine security, incompatibilites might arise due to operating systems or web browser versions, and the network infrastructure between the voter and the central election system might be compromised with a man-in-the-middle attack.

4.4.4 Usability

Traditional election systems struggle with usability. Verifiable systems add more steps and complexity, making usability even more difficult. The mechanics of marking a ballot become more complex with code voting as in Remotegrity, and position or shape matching as in Prêt à Voter and Punchscan. Individual verification, not even possible in traditional systems, is an entirely new process that voters must master to take full advantage of E2E guarantees.

In 2014, a team of researchers from Rice University undertook a quantitative, experimental study of the usability of Helios, Prêt à Voter, and Scantegrity II [2]. They aimed to quantify usability using the ISO 9241-11 standard axes of effectiveness, efficiency, and satisfaction. Their results show that these systems broadly fail on these axes even for typical voters who are uninterested in performing additional verification steps.

The Rice study found the systems were not effective as significant number of voters failed to cast a ballot with each system. Troublingly, many of those voters thought they had in fact successfully cast a ballot; in a real election they would have left the voting process unfinished without even knowing to ask a poll worker for assistance. By contrast, traditional systems have near-100% success rates [12].

The systems also lacked efficiency, as they all required significantly more time – almost twice as long – to complete as a traditional system.

The usability of an election system is crucial for that system to not disenfranchise voters, and for voters to generally have confidence in the election results. The Rice study shows that adding E2E guarantees can be a Pyhrric victory when the resulting system is unusable for non-expert voters.

4.4.5 Accessibility

There are ability requirements for many E2E systems in various stages of the voting process. For example, a sighted voter is able to see the correspondence between candidate position and marking position on a Punchscan ballot, but a non-sighted voter cannot without assistance. In addition to obstacles to marking a ballot, some schemes with individual verification lack provisions for disabled voters to participate in individual verification without assistance. Information required for verification is frequently delivered through a paper receipt, an invisible ink code, or requires writing down receipt data.

Accessible verification protocols have been proposed that take care to protect voter secrecy and allow participation in individual verification [18]. However, these protocols require using accessibility equipment with an audio, sip-puff, or switch interface to read and mark the unencrypted ballot. The device must therefore be trusted not to record the votes, which would violate voter secrecy. The device must also represent the ballot faithfully to the voter so that votes are recorded as intended.

Requiring trust in assistive devices is not unique to E2E systems [48]. In non-E2E systems, though, trust is already widely distributed. In the context of having to trust the chain of custody of ballots, the integrity of poll workers, and the outcomes of any audits, having to trust an assistive device is a relatively small concession to make in an already-flawed system.

On the other hand, a well-designed E2E system requires a much smaller base of trust for voters to have confidence in the results of an election. By requiring an expanded base of trust in order to be accessible, the existing E2E systems undermine their E2E properties.

4.4.6   Social & Political

Novel election systems face a difficult bootstrapping problem: in order to be adopted in large-scale elections, they must have a successful track record. However in order to build up that track record, systems must be successful despite the limited resources available during small-scale pilot programs. With limited resources, corners are cut in the implementation of the election system leading to a greater chance that problems with equipment, software, and support will undermine confidence in the system.

This confidence in election systems generally, and E2E systems in particular, is fragile in the eyes of the public. When election systems fail during an election or are revealed to have substantial integrity issues, the perception of all similar systems is tainted, no matter the differences between specific systems or the reassurance of E2E guarantees. Failure of a legacy computerized system can poison the well and make the public reject a novel system by association.

For example, The Federal Constitutional Court of Germany issued a decision in 2009 in the wake of a hacking demonstration on electronic voting machines used in previous elections [28]. They decided that electronic systems may only be used in elections if "the result can be examined reliably and without any specialist knowledge of the subject", a standard which E2E systems have not been able to meet in practice [12]. Similarly after reports critical of RIES, a popular movement successfully advocated for a ban on Internet voting in the Netherlands.

Broader computer security concerns are becoming topics of household conversation with vulnerabilities like Heartbleed and droves of personal data compromises making the headlines. These concerns rightly make the public wary of any system with a computerized component, even if the Internet is not involved. The challenge for E2E systems is to overcome this broader skepticism by demonstrating integrity in a way accessible to non-experts without making it more difficult to vote.

Chapter 5

Required Properties of E2E Systems (Dan) (100%)

In August 2010, the U.S. Election Assistance Commission issued a set of testing requirements for UOCAVA remote electronic voting system pilot projects [58]. The general categories of requirements specified by the EAC included functional requirements, such as the need for the system to produce paper records of voter choices and generate human-readable ballot images; requirements on software development, such as allowable programming languages and coding conventions; usability, accessibility and privacy requirements, such as that a voter's ballot choices must remain private and that provisions must be made to support voters with disabilities; security requirements, including logging requirements, requirements on communications security within the system, and requirements on physical security and penetration resistance; quality assurance requirements describing the testing that must be done on the systems; and requirements about configuration management mechanisms, technical information, and documentation to be provided by system vendors.

The EAC have some serious shortcomings, one of which is that several of the requirements seem arbitrary. For example, they specify (in Section 2.1 of the requirements document) that the voting system shall achieve a target error rate of no more than one in 10,000,000 ballot positions, with a maximum acceptable error rate in the test process of one in 500,000 ballot positions, without any justification for those numbers. They further specify (in Sections 2.1.1.1–2) that "memory hardware, such as semiconductor devices and magnetic storage media, shall be accurate" and that "the design of equipment in all voting systems shall provide for protection against mechanical, thermal, and electromagnetic stresses that impact voting system accuracy" without any guidance on how to evaluate such accuracy or protective ability.

In addition to these shortcomings, some of the EAC requirements are inappropriate or invalid. The most obvious example of this is the set of requirements that mandate specific "structured programming" characteristics of software implementation languages (Sections 4.1 and 4.4), which seem to eliminate functional programming languages such as Haskell and Erlang—widely used in implementing high-assurance systems—from consideration entirely.

If these issues were addressed, the EAC requirements could serve as a solid baseline set of requirements for remote electronic voting systems; effectively, addressing the "IV" in "E2E VIV". However, they are not strong enough to guarantee end-to-end verifiability, which—as previously discussed—is essential when considering Internet voting systems for use in real elections. Thus, we describe here a set of required properties for E2E VIV systems that has significant overlap with the EAC requirements.

The set of E2E VIV requirements can be broadly divided into two groups: *technical requirements* and *non-functional requirements*. Technical requirements are those that can be directly addressed by the design and implementation of the system, such as authentication requirements for voters and election officials. Non-functional requirements are those that are imposed on the system by external entities or where the system depends on external behaviors outside its control, such as specific election certification guidelines and operational procedures. Each of these groups is itself divided into several categories, and Figure 5.1 gives a high-level overview of these.
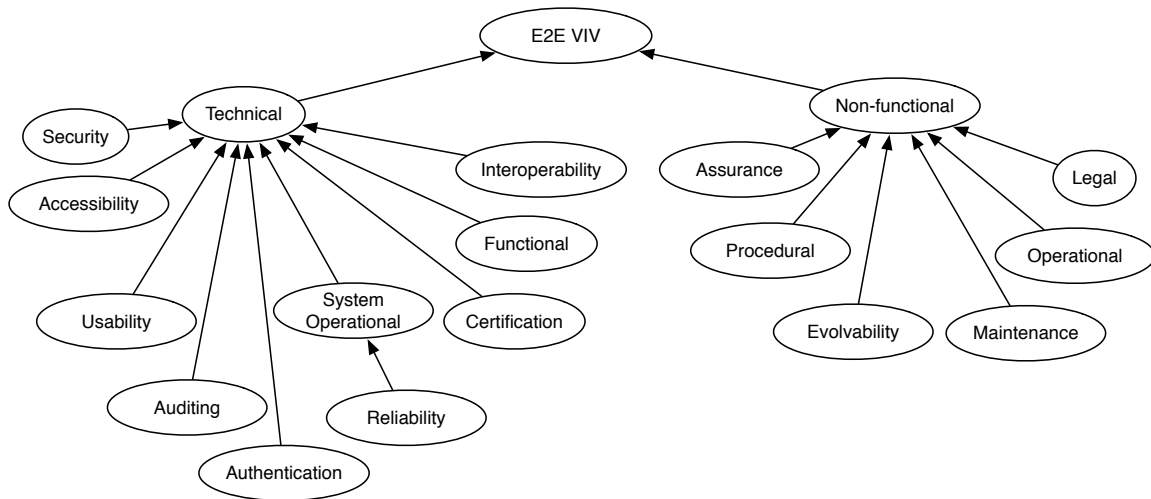
Figure 5.1: The hierarchy of requirements for E2E VIV systems.

The following is a high-level description of the categories and many of the requirements within each; Appendix A contains a complete listing of all E2E VIV system requirements expressed in the Business Object Notation.

5.1 Technical Requirements

There are ten categories of technical requirements for E2E VIV systems: functional, accessibility, usability, security, authentication, auditing, system operational, reliability, interoperability, and certification.

5.1.1 Functional

The functional requirements of an E2E VIV system deal primarily with the casting and recording of ballots and associated voter records. One important requirement is that there must be a correspondence between the recorded ballots and the voters that are listed as having voted; a ballot cannot be recorded without a voter casting it, and a voter cannot be listed as having voted without casting a ballot. Similarly, if a voter is informed by the system that her ballot has been successfully cast, the system must correctly retain the record of her having voted and her cast ballot information even in the event of server failures.

Another functional requirement is the property of *receipt freedom*: it must be impossible for a voter to prove to anybody any information regarding how she voted her ballot, beyond what can be mathematically deduced from the final distribution of votes. For example, if a referendum passes with 100% of the vote, there is no way to hide the fact that every voter approved of the referendum; however, if the result is mixed, it must be impossible for any individual voter to prove how she voted. This must be the case even when the voter can create digital evidence of her actions by, for example, video recording the ballot casting process or photographing a completed ballot.

In some elections voters are allowed to cast multiple ballots with only the last cast ballot counting toward the final election tally, while in others voters are prohibited from casting multiple ballots. The system must accommodate both of these election formats, ensuring that only the last cast ballot is counted for each voter when multiple ballots are allowed and ensuring that each voter casts at most one ballot otherwise.

Maintaining voter anonymity is critical, so it must be impossible after the election to reconstruct a link between a cast ballot and any identifying information about the voter who cast it. However, in systems that support the casting of multiple ballots, it is important to maintain links between voters and their ballots *during* the election to ensure that later ballots replace the correct earlier ballots. To balance these concerns, any link between a ballot and the voter who cast it must be irrevocably broken once it is conclusively determined that the ballot will be counted toward the final tally.

Finally, because the voter should be able to focus on the voting process without undue distractions or external influences, the voting system must not display or permit the display of any advertising or commercial logos during a voting session; the exception to this rule is that an election jurisdiction may display its own logo to the voter during the voting process. Along the same lines, the voting system must not display any links to other Internet sites outside of the voting system, except to provide help with the actual mechanics of voting.

5.1.2 Usability

The usability of an E2E VIV system is critical to its successful adoption and use. Since the user experience is so important, many of the requirements of the system have some relation to usability even though they may be categorized under other headings. There are, however, two requirements that are exclusively related to the usability of the system with respect to vote casting and one general usability requirement that applies to the system as a whole.

The first vote casting requirement is that, if a voter receives a final vote confirmation (e.g., "Thank you for voting!" or a similar notice) from the system, she can be certain that her ballot was recorded correctly. This is the usability counterpart to the functional requirement that ballot records and voter records must be maintained correctly even in the event of server failures.

The second vote casting requirement is that, if a voter is uncertain whether or not her ballot was recorded (e.g., she clicked a "submit" button but never got a response from the system), she must be free to attempt to vote again.

Finally, usability testing must be performed on any E2E VIV system before it is deployed. The reports of the usability testing must be made public, and the system must achieve satisfactory test results before being deployed in a real election.

5.1.3 Accessibility

Accessibility—the property of being usable by and useful to the disabled—is one of the main goals of an E2E VIV system. It is closely related to usability, but there are several requirements associated specifically with accessibility that go beyond typical usability requirements.

Users must be involved in the design of the system to identify accessibility constraints at each stage of the development process. Consideration must be given to the system's compatibility with existing technologies designed to help disabled individuals; for example, the system should be developed in a way that allows assistive input devices such as switches, eye trackers and screen readers to be used in addition to keyboards, mice and touchscreens. Similarly, the system's presentation of voting options should be optimized to voters' needs by providing alternative display fonts, audio representations, braille representations, and other representations as appropriate.

All possible measures must be taken to ensure that the system can be used by all voters and, if that is not possible in all circumstances, to provide access to alternative methods of voting for those voters who cannot use the system.

Finally, accessibility testing must be performed in addition to the previously-mentioned mandatory usability testing. The reports of the accessibility testing must be made public, and the system must achieve satisfactory test results before being deployed in a real election.

5.1.4 Security and Authentication

Security and authentication are closely related and together represent the broadest set of technical requirements, consisting of both requirements on the E2E VIV system itself (data storage, communications, etc.) and requirements on the voting and counting processes enabled by the system (voter authorization, voter privacy, tally accuracy, etc.).

It is crucial that data integrity be ensured throughout the system. Therefore, measures must be taken to ensure that no data can be permanently lost in the event of a breakdown or fault affecting the system; that the system maintains the integrity of the voters' register, lists of candidates, ballot information, cast ballots, and other critical information, in addition to authenticating the original source(s) of that information and tracking provenance where appropriate; that all data communications within the system have associated integrity checks; that system equipment under the control of the electoral authority is protected against influences that could modify the election results; and that the integrity of the election results does not depend in any way upon the security of system equipment not under control of the electoral authority. The system must perform regular "health checks" to ensure that data integrity has been maintained, that all its components are operating in accordance with their specifications, and that all system services are available.

Accurate timing information is critical to security, both in terms of providing evidence of compliance with applicable regulations and in terms of detecting attacks on and potential breaches of the system. The system must therefore maintain reliable synchronized time sources, with sufficient accuracy to maintain timing data for audit trails, election observation data, and time limits for various aspects of the election process. It must be possible to determine, using the timing information stored by the system, whether nominations (and, if required, acceptance thereof by the candidate or electoral authority), voter registration, and vote casting have occurred within the prescribed time limits for those actions.

Authentication and authorization are also important aspects of security. The system must ensure that each individual can be identified uniquely, so that there is no possibility of mistaking one individual for another. The system must also maintain the privacy of individuals, by ensuring that all personally identifiable data is kept confidential as far as is allowed by the legal requirements of the electoral jurisdiction. The system must allow access to each of its services only to authorized users; for example, only individuals who represent the electoral authority may be allowed to load ballot information into the system.

The authentication mechanisms used to gain access to the system must, as far as possible, protect authentication secrets (passwords, one-time access codes, biometrics, etc.) so that unauthorized entities cannot acquire them. Authentication to the system may not be carried out through third parties; that is, existing online accounts such as those at Facebook, Google and Twitter may not be used as authentication mechanisms. The security of the authentication mechanism must not be affected by any potential breach of any public or commercial database (e.g., a credit card database, the Social Security database), and it should not be possible for an attacker to impersonate a voter even if the entire database used for authentication in the system is compromised. Individual authentication secrets themselves must be changeable or revokable at any time, at the behest of either the individual or election officials, and must be changed for all individuals at least once in every election cycle.

With respect to the actual voting process, only eligible voters may be allowed to cast ballots and the system must ensure that only the appropriate number of ballots is cast by each voter. It must be possible for a voter to verify that the system has presented her with an authentic ballot and, in the case of remote voting, that she has a secure connection to an official server.

The privacy of the vote must be preserved end-to-end to the maximum extent possible, and individual voters may not waive the privacy of their votes. In the case of remote voting, vote privacy must be preserved even in the presence of arbitrary malicious code on the voter's computer (corrupted client software, key logging software or devices, etc.). Any client software used in remote voting must not send data to any Internet host except those associated with the E2E VIV system or provide any information to third parties (e.g., Facebook, Twitter, etc.) regarding the act of voting. Any residual information that could be used to discover a voter's choices must be destroyed after a ballot has been cast; if a voter uses a computer outside the control of the electoral authority to cast her vote, she must be provided with instructions for destroying any such information on that computer.

With respect to vote counting, the system must accurately count the votes and the counting process must be reproducible. The system must also maintain the availability and integrity of all information used to generate the final tally and all information regarding the counting process itself for as long as required. Vote tabulation must be *software independent*; it must be possible to reconstruct a correct tally from some record even if the election system software is compromised.

Finally, it is expected that a deployed E2E VIV system will be an attractive target for highly-capable adversaries that wish to influence election results or to disrupt election processes. With this in mind, the system must be designed and tested assuming that an adversary has a budget of US$10 per voter per election that can be applied toward any critical subset of votes or voters of their choosing; thus, an E2E VIV system for use in a U.S. presidential election would need to be designed and tested assuming that an adversary has a budget of approximately US$1,300,000,000.

The electoral authority shall have overall responsibility for compliance with these security requirements, and such compliance shall be assessed by independent bodies as appropriate.

5.1.5 Auditing

The ability to perform comprehensive audits of system activity is one of the important distinguishing aspects of an E2E VIV system as compared to other voting systems; as a result, there are several system requirements related specifically to auditing, in addition to those security requirements (such as the tracking of accurate timing information) that touch on auditing.

First, the audit system must be designed and implemented as part of the E2E VIV system from the beginning; it cannot be added as an afterthought to an existing system. Audit and monitoring facilities must be integrated into all levels of the system, from low-level communications among individual computers to high-level interactions with election officials. The system must keep audit logs of all activity relevant to the conduct and outcome of the election, and these logs must be unmodifiable once they are written and as complete as possible without violating voter privacy.

The audit system must actively report on potential issues and threats, rather than merely serving as a passive repository of system logs. It must record at least the following events and actions with accurate timing information: all voting-related information, including the number of eligible voters and votes cast, the number of invalid votes, count and recount results, etc.; any detected attacks on the operation of the system or its communication infrastructure; and any system failures, malfunctions, or other detected threats to proper system operation. It must provide sufficient information to election observers in real time, and after the election's conclusion, to verify that the election is carried out in accordance with applicable law.

The audit system must also be able to cross-check and verify the correct operation of the voting system and the accuracy of the election results, to detect voter fraud, and to prove that all counted votes are legitimate and that all ballots have been counted. In situations where the system cannot verify the legitimacy of all the votes, it must be capable of giving an upper bound on the number of affected ballots. If a tradeoff must be made between maintaining voter privacy and identifying the perpetrators of fraud, the system must resolve that tradeoff in favor of voter privacy.

In order for an E2E VIV system to be trusted, its auditability must extend to its own source code as well as the activities it performs during an election. Therefore, the E2E VIV system software, including any official monitoring and auditing applications, must be published in source form along with documentation, instructions for building and running, and a digital signature as a proof of authenticity.

5.1.6 System Operational

System operational requirements ensure that the system is configured, updated, and run in a transparent, accountable way that allows for the other requirements to be fulfilled. One important such requirement is that there must be official published manifests of the system used to run any election, indicating details of the software and versions used, dates of installation, and brief descriptions of their functionality. Both public and private manifests must be maintained;

these should be identical, except that details about software used solely to protect the system against attacks may be omitted from the public manifest for security reasons. Well-defined procedures must exist for both updating the manifests to reflect changes to the installed software and checking the installed software against the manifests to detect tampering.

Before every election period, all equipment (including all software) must be checked and approved in accordance with procedures devised by the electoral authority. This check must include a check of the software against the manifests, as well as any necessary tests to establish that the system complies with its technical specification.

During an election period, key equipment must be located in a guarded, secure area at all times. There must be a contingency plan for system failures including provisions for backup and failover systems, which must conform to the same standards and requirements as the systems they replace. In addition, sufficient arrangements for data backup must be in place, continuously monitored, and always available during the election; election staff must be ready to intervene rapidly, according to a procedure established by the electoral authority, in the event of incidents during an election. Individuals responsible for the voting equipment must follow established procedures to ensure that the equipment and its use satisfy requirements.

To ensure accountability on the part of the electoral authority and election system vendors, a report containing every software manifest change and every violation of data security, system security, physical security or control procedures must be prepared and made public by the electoral authority within a reasonable amount of time after every election.

5.1.7 Reliability

In order to be successfully used to conduct elections, an E2E VIV system must satisfy strict reliability requirements with respect to both its behavior under normal conditions and its behavior while under attack.

In general, the back-end (i.e., non-voter-facing) components of the system must have a proven mean time before failure (MTBF) of at least one week under constant peak expected load; that is, it must have been shown in multiple actual tests of mock elections to run continuously for at least a week at the highest expected voter participation rate. The one week MTBF requirement applies only during normal operation, not while the system is under attack.

In addition to the MTBF requirement, the system must also exhibit 99.9% uptime during the election period, and must be able to recover from any failure other than a regional natural disaster or malicious attack in less than 10 minutes. This must be demonstrated by inducing failures in actual mock election situations, e.g., by unexpectedly unplugging servers or disconnecting storage devices. Redundant failover components must be in place for all critical components of the system in order to ensure the 10 minute maximum recovery time.

An E2E VIV system is likely to be a tempting target for distributed denial of service (DDoS) attacks; it must be able to continue correct operation during a sustained DDoS attack at a specified level on any combination of its back-end components with no more than a specified acceptable degradation of response time to voters during the attack. The specified attack level and acceptable degradation of response time will vary among election types; for example, a system running a national election must be able to resist a significantly higher level of attack than a system running a county election. Our initial suggestions for the thresholds for a national election are that the system must continue operating correctly under a DDoS attack at a level of 100 gigabits per second, with no more than a 15 second degradation of response time.

The ability of the system to survive DDoS attacks and continue operation while fulfilling the response time requirements must be demonstrated in the actual network configuration to be used during the election, and the required thresholds for these values should be re-evaluated every election cycle to keep pace with advancement in attack technology.

5.1.8 Interoperability

E2E VIV systems must use open, rather than proprietary, data and communication standards for interoperability among their various components and services. Whenever possible, the Election Markup Language (EML) or a similar standard ratified by an international standards body should be used for data interchange and configuration within the system. The standards used within the system should allow for localization of election data in situations where such localization is required.

The log data for the system, and documentation describing its meaning and format, must be available for public download so that anybody can download, inspect, and publish concerns based on the system logs.

5.1.9 Certification

In order to provide sufficient evidence for certification of an E2E VIV system, each functional requirement must have an associated set of automated tests that demonstrate its fulfillment. These tests must be runnable on demand, and their results should be unambiguous and easily understandable.

In addition, the election protocol implemented by the system (communication, cryptographic, etc.) must have associated formal proofs of correctness and security.

5.2 Non-functional Requirements

There are five categories of non-functional requirements for E2E VIV systems: operational, procedural, legal, assurance, and maintenance/evolvability.

5.2.1 Operational

The operational requirements on E2E VIV systems deal with several distinct issues including election and registration timing, voter registration, candidate nominations and lists, receipt freedom, voter assistance, and the handling of hardware and software platform issues and election integrity violations.

Voters must be informed, in clear and simple language, of how electronic voting will be organized and what steps a voter will need to take in order to participate and vote electronically. Support and guidance with respect to voting procedures must be available to all voters. In the case of remote voting, such support and guidance must be available through a different, widely-available communication channel (such as a dedicated phone number) in addition to being available via the Internet. Voters must receive clear guidance about exactly what client configurations (i.e., hardware platforms, operating systems, browsers, browser plugins, other applications, and versions thereof) are required by or supported by the E2E VIV system, and what common components, plugins, or other software (e.g., pop-up blockers, script blockers) may interfere with voting. In addition, voters must receive clear guidance about configuration choices they can make to more strongly protect their privacy; for example, disabling cookies and browser history logging, running privacy-protecting browser plugins, voting from temporary virtual machines, logging out of social networks, disabling non-election-related Internet communications, etc.

In any election carried out using an E2E VIV system, the relevant jurisdiction's legal provisions must provide for clear timetables concerning all stages of the election. The period during which a vote may be cast electronically must not begin before the public is notified of the election; in particular, with respect to jurisdictions that allow remote electronic voting, the voting period must be defined and made known to the public well in advance of its start. In jurisdictions where remote voting takes place concurrently with voting at supervised polling stations, the time periods for remote and supervised voting need not be identical; however, remote voting should not be allowed after the period for supervised voting has ended.

An E2E VIV system must have a publicly accessible voters' register that is regularly updated. Each voter must be able to check, at a minimum, that her information as recorded on the register is accurate, and must be able to request corrections of any inaccurate information. In jurisdictions where remote electronic voting takes place concurrently with voting at supervised polling stations, the system must be designed in a way such that it prevents any voter from voting more than once.

On any electronic ballot, all voting options must be presented equally; that is, there must be no distinguishing fonts, sizes, styles, or other embellishments that could cause one or more of the voting options to be perceived by a voter as "preferred". The ballot must be free of any information about the voting options—biographical information about candidates, interpretations of and statements about ballot initiatives, etc.—other than information strictly required for casting the vote or required by law to be on the ballot (for example, candidate party affiliation is often required to appear). The system must also avoid displaying any messages that may influence voters' choices. Additional information about voting options might be made available from an electronic voting site as part of an E2E VIV system, separate from the actual electronic ballot; if so, such information must be presented without bias.

E2E VIV systems are likely to be made available for testing by voters and election officials, both before and during elections. They must therefore indicate clearly, before the final casting of any ballot, whether the ballot is being cast in a real election or as part of a test. In the case of a test that occurs simultaneously with a real election, individuals casting test ballots should subsequently be directed to the appropriate voting channel for casting real ballots.

E2E VIV systems must exhibit receipt freedom (mentioned previously in the technical requirements); that is, they must not enable the voter to possess a proof of the choices they have made in a cast vote. In a supervised environment, voting information should disappear from the display (visual, audio or tactile, depending on accessibility requirements) used by the voter to cast the vote as soon as the vote has been cast. When a paper proof of an electronic vote is provided to the voter at a polling station, the voter must not be allowed to show it to any other person or to remove it from the polling station.

With respect to counting the votes, an E2E VIV system must not allow the disclosure of any vote counts until after the system has stopped accepting electronic ballots. Tally information must not be disclosed to the public until after the end of the voting period (including all polling station voting). Any decoding required for the counting of the votes shall be carried out as soon as practicable after the end of the voting period; representatives of the electoral authority must be able to participate in, and observers must be able to observe, the counting process. A record of the counting process must be kept, including timing information and identifying information for all persons involved in the counting process. In the event of any irregularity affecting the integrity of votes, it must be recorded that the affected votes had their integrity violated; the effect of such integrity violations on the election results will vary based on the legal provisions of the involved jurisdictions.

Finally, any deployed E2E VIV system must function correctly as an open system, where large parts (specifically, any remote client hardware and software) are unknown, unsecured, uncertified, and completely out of the control of election officials. The system must be auditable to the extent possible given this requirement, and the conclusions drawn from the audit process should be applied in future elections.

5.2.2 Procedural

Successful deployment of E2E VIV systems requires certain procedures to be followed with respect to their provisioning, certification, maintenance, availability, and use. Because such systems are critical pieces of public infrastructure, information about their functioning must be publicly available and information about the specific components of a system must be disclosed, at least to the relevant electoral authority, as required for verification and certification purposes. Before any such system is introduced, at appropriate intervals after its introduction, and in particular when any changes are made to the system, an independent body appointed by the electoral authority must verify that the system is working correctly and that all necessary security measures have been taken.

After introducing a system, the electoral authority must take steps to ensure that voters undesrtand its use and have confidence in the system; these may include outreach, practice elections, and any other measures the electoral authority sees fit. In particular, voters must be given an opportunity to practice any new electronic ballot casting method before, and separately from, the casting of an electronic ballot during a real election.

The electoral authority must take steps to ensure the reliability and security of the E2E VIV system; for example, guarding equipment, providing suitable reliable power supplies, etc. All possible steps should be taken to avoid the possibility of fraud or unauthorized intervention during the voting process, and the electoral authority must satisfy itself that the E2E VIV system is genuine and operates correctly before using it to conduct a real election.

Only individuals appointed by the electoral authority should have access to the central infrastructure, the servers, and the election data, and clear rules should be established for such appointments. Critical technical activities must be carried out by teams of at least two people, and the composition of such teams must be regularly changed. As far as possible, critical technical activities should take place outside of election periods.

Observers must be allowed to be present, to the extent permitted by law, to observe and comment on the conduct and establishment of the results of any election conducted using an E2E VIV system. During an election period, any authorized intervention affecting the system must be carried out by a team of at least two people, be the subject of a written report, and be monitored by representatives of the election authority and election observers.

The system must maintain the availability, integrity, and confidentiality of the votes. It must also keep the votes sealed until the counting process begins. Any votes stored or communicated outside controlled environments must be encrypted. Recounts must be possible, and any features of the system that may influence the correctness of the result must be verifiable. The system must also support partial or complete re-runs of elections.

Finally, there must be clear technical and legal procedures to be followed in the event that voters can prove that their votes were not received accurately or counted, or in the event that the official election verification application does not verify that the results of the Internet portion of the election are correct.

5.2.3 Legal

Legal requirements arise primarily from the application of existing law to E2E VIV systems. These include requirements on accessibility and availability; on the counting of votes, number of votes per voter, and anonymity of votes; and on restrictions with respect to reverse engineering or testing of E2E VIV systems.

To comply with accessibility and availability requirements, the voting interface of an E2E VIV system must be understandable and easily usable, and registration requirements for electronic voting must not pose an impediment to voter participation. E2E VIV systems should be designed, as far as is practicable, to maximize the opportunities they provide for the disabled. Unless remote electronic voting channels are universally accessible, they must be used only as an additional and optional means of voting beyond polling places or more traditional remote voting methods.

The E2E VIV system must insure that at most one electronic vote from each voter is included in the final tally, that every vote cast electronically is counted, and that each vote cast electronically is counted only once. In jurisdictions where electronic and traditional voting channels are used in the same election, there must be a secure and reliable method to aggregate all votes, prevent multiple votes by the same voter from being counted, and calculate correct results.

The way in which voters are guided through the process of electronic voting should be designed to prevent their voting precipitately or without reflection. Voters must be able to alter their choices at any point during an electronic voting process before casting their vote, or to stop the voting process, without their previous choices being recorded or made available to any other person under any circumstances. The electronic voting system must not permit any manipulative influence to be exercised over the voter during the voting process, must provide the voter with a means of participating in the election without exercising a preference (e.g., by casting a blank ballot), must indicate clearly to the voter when the voting procedure has been completed, and must preserve voter anonymity.

There must be no legal impediments to interested parties who want to study the E2E VIV system. In particular, no nondisclosure agreement or contract of any kind may be required for such download and study, or for building, testing and publishing test results for the E2E VIV system.

5.2.4 Assurance

There are several assurance requirements with respect to the implementation, documentation, and licensing of E2E VIV systems. First, client side software—that is, any software that is expected to be used on a system serving as a voting terminal, whether a supervised machine at a polling place or an unsupervised machine belonging to a voter— must be free of known bugs on a wide range of platform and software stack combinations. As previously discussed in Section 5.2.1, the specific supported platform and software stack combinations for the software must be clearly conveyed to voters. The system must exhibit strong security with respect to voter authentication, such that there is no way to automate forging or invalidation of voter authentication credentials without compromising the cryptographic protocols or secrets used in the system.

All aspects of the design, architecture, algorithms and documentation for the entire Internet voting system (not just the E2EV core) should be published and available for free download by anyone. As the system changes, all associated documentation must be kept up to date, and no new version of an E2E VIV system should be certified until it has up-to-date documentation.

The source code, build scripts, issue tracking system, security features, and related development information for the entire Internet voting system—all versions, for all supported platforms—should be made publicly available for free download and inspection, under a license that permits anyone to download, build, instrument, and test the system.

5.2.5 Maintenance and Evolvability

Maintenance and evolvability requirements are closely related, and essentially stipulate that an electoral authority, or any entity engaged by an electoral authority, must be able to change an E2E VIV system in response to changes in the legal or technical environment in which it operates.

The electoral authority must have the right and the ability to update the election system to conform to changes in applicable law, available technology, or threats to system integrity independent of the original vendors of the system. The electoral authority must also have the right and ability to patch election systems to correct flaws discovered in the algorithms, implementation, or deployment, subject to the documentation update requirement described above and the procedural requirement that the system must be re-verified for correct operation before being used to conduct a real election.

Chapter 6

Crypto Specification (Aggelos et al.) (15%)

6.1 Ideal Functionality of an E2E System—$\mathcal{F}_{\text{e2e}}$

The basic protocol followed by any system implementing E2E verifiable elections can be characterized by an *ideal functionality*. This ideal functionality, called $\mathcal{F}_{\text{e2e}}$ and presented in Figure 6.1, recognizes and interacts with the election authority EA, the set of eligible voters $V_1, \ldots, V_n$ and the auditor AU. These are "ideal-world" entities; a "real-world" implementation of $\mathcal{F}_{\text{e2e}}$ may involve more parties that will enable the implementation to realize the ideal functionality.

$\mathcal{F}_{\text{e2e}}$ accepts a number of commands from the election authority EA, the voters and the auditors. At the same time it informs the (ideal world) adversary of certain actions that take place and is influenced by the adversary to perform certain actions. The ideal functionality keeps track of which parties are corrupted and may act according to their corruption status.

$\mathcal{F}_{\text{e2e}}$ has two parameters:

1. A function $f : (X \cup \{\bot\})^n \to E$ that defines the election function, where X defines the set of all possible ways for an individual voter to vote and E is the set of all possible election results. The notation X^n denotes all possible strings of length n over the alphabet X. The symbol $\bot$ stands for "undefined." The election function f is invariant with respect to $\bot$, i.e., $f(\bot, x) = f(x)$ for all x.

2. A relation Q that defines the level of sensitivity to manipulation permitted by $\mathcal{F}_{\text{e2e}}$. In particular, for two possible election results T and T' we say that $Q(T, T')$ holds if and only if T' is sufficiently close to T. For the most strict version of $\mathcal{F}_{\text{e2e}}$ we define Q to be the equality relation over E (that is, $Q(T, T')$ holds if and only if $T = T'$).

The ideal functionality $\mathcal{F}_{\text{e2e}}^{f,Q}$ captures the following set of security characteristics:

- Provided the EA is not corrupted, the adversary is incapable of extracting the voters' selections.

- Provided the EA is not corrupted, all votes are recorded and tallied according to the election function $f(\cdot)$.

- Even if the EA is corrupted, a set of well defined votes are assigned to the voters of the election (however, such votes may deviate from the original voters' intent). The votes cannot be manipulated when the EA is honest.

- Even if the EA is corrupted, the functionality consistently returns the same tally result to all parties that request it. Furthermore, the functionality always tests the reported tally according to the predicate Q and reports the outcome, hence any substantial (according to Q) deviation from the recorded tally will be detectable by all honest parties.

> 13: Dan parked this here, because it doesn't belong in the "require-ments" chapter, but it may not really belong here either. Still, it or some variant of it probably does. Do with it what you will...

Functionality $\mathcal{F}_{\text{e2e}}^{f,Q}$

The functionality recognizes and interacts with the following parties: (i) the election authority EA; (ii) the eligible voters $\mathcal{V} = \{V_1, \ldots, V_n\}$; (iii) the auditor AU; and (iv) the adversary $\mathcal{A}$. It is parameterized by the relation Q over E and the election function $f : (X \cup \{\bot\})^n \to E$.

- Upon receiving an input (Create, sid, B) from the EA, record the tuple (sid, B) such that sid is the election identifier and B is a string defining the ballot of the election. Send (Create, sid, B) to the adversary $\mathcal{A}$.
- Upon receiving an input (Deliver, sid) from EA, deliver (B, s_i) to each voter V_i, where s_i is some voter-specific information that is provided by the adversary $\mathcal{A}$.[a]
- Upon receiving an input (Vote, sid, a) from V_i, select a unique identifier vid and record the tuple (vid, a) provided $a \in X$. If EA is honest, send (Vote, sid, vid) to the adversary $\mathcal{A}$; if EA is corrupted, send (Vote, sid, vid, V_i, a) to $\mathcal{A}$.[b]
- Upon receiving (RecordVote, sid, vid, b) from $\mathcal{A}$, verify that a tuple (vid, a) has been previously recorded and then record the tuple (V_i, a, b) provided that (i) V_i is a voter that has not previously been assigned a vote,[c] (ii) the value a is a valid choice consistent with the ballot description B, and (iii) the value b is unique amongst received RecordVote messages. Finally, return (Receipt, b) to V_i.
- Upon receiving an input (Tally, sid) from the EA, collect all recorded inputs $\{(V_j, a_j, b_j)\}_{j \in \tilde{\mathcal{V}}}$, where $\tilde{\mathcal{V}}$ is the set of voters that voted successfully, and set $a_j = \bot$ for all $j \notin \tilde{\mathcal{V}}$. Compute $T = f(\langle a_1, \ldots, a_n \rangle)$ and return (Tally, T) to $\mathcal{A}$.
- Upon receiving (RecordTally, $sid, \mathcal{M}, \hat{T}$) from $\mathcal{A}$, where $\mathcal{M}$ can be parsed as a polynomial-size circuit, set $\langle a'_1, \ldots, a'_n \rangle = \mathcal{M}(a_1, \ldots, a_n)$ and if $\exists j : (a'_j \neq a_j)$ and EA is honest then ignore the message. In any other case, record (Result, $T', \hat{T}$) and $\langle a'_1, \ldots, a'_n \rangle$, where T' is the election result calculated as $T' = f(\langle a'_1, \ldots, a'_n \rangle)$.
- Upon receiving (ReadTally, sid) from any party, return (Result, $\hat{T}, Q(\hat{T}, T')$).
- Upon receiving (Audit, b) from from any party, recover the triple (V_j, a_j, b_j) such that $b_j = b$ and return 1 if and only if $(a'_j = a_j)$ and 0 otherwise.

---

[a]In some systems, voters may request this information actively; hence, $\mathcal{F}_{\text{e2e}}$ will be passive and will not deliver the ballots. In such cases the adversary will adaptively provide the s_i values.

[b]In some systems the voter identity V_i can be leaked to the adversary during ballot casting.

[c]In some systems the voter is allowed to change his/her mind and hence vote multiple times.

Figure 6.1: The ideal functionality $\mathcal{F}_{\text{e2e}}^{f,Q}$.

- The functionality preserves voter intent and, in case of vote manipulation, the voter or an auditor can use the unique receipts provided in the completion of ballot-casting to test whether voter intent was manipulated by a corrupt EA. Any party may use those receipts, hence verification is "delegatable."

Consider now a protocol π that implements syntactically the ideal functionality $\mathcal{F}_{\text{e2e}}$ (i.e., has the same I/O characteristics as $\mathcal{F}_{\text{e2e}}$). Following standard notation and terminology we have the following:

Definition 6.1.1 *Let f be an election function and Q a predicate over the election results. The protocol π implements $\mathcal{F}_{\text{e2e}}^{f,Q}$ provided that, for all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}$ so that for all environments $\mathcal{Z}$ it holds that*

$$\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{Exec}_{\mathcal{S},\mathcal{Z}}^{\mathcal{F}_{\text{e2e}}^{f,Q}} \ .$$

Note that the protocols that will be considered in practice may utilize simpler ideal functionalities. In such cases, the protocol π implements $\mathcal{F}_{\text{e2e}}$ conditional on the existence and availability of these other functionalities. Such functionalities include "authenticated channels", a "write-only bulletin board", etc.

Party corruption. As stated, the ideal functionality $\mathcal{F}_{\text{e2e}}$ enables the adversary $\mathcal{A}$ to corrupt parties by issuing special $(\texttt{Corrupt}, P)$ messages. Given such a message, the ideal functionality $\mathcal{F}_{\text{e2e}}$ will divulge to the adversary the complete I/O transcript from the interface between $\mathcal{F}_{\text{e2e}}$ and P. We distinguish between static and adaptive corruptions. In the case of static corruptions all messages $(\texttt{Corrupt}, P)$ are delivered at the onset of the execution, while for adaptive corruptions they can be delivered at any time. For brevity we do not explicitly include the actions taken for $\texttt{Corrupt}$ messages in the description of the functionality.

In the real world, the corruption of an entity expresses an action taken by the adversary that results in the complete control of the entity's computing environment. A corrupted voter, specifically, loses privacy completely and the adversary may take any action on her behalf. For example, if corruption of a voter happens prior to ballot casting the adversary may vote on her behalf, while if corruption of a voter happens after ballot casting the adversary will learn her choice. If the adversary corrupts the EA, it may try to manipulate some voters' ballots however to the extent permitted by $\mathcal{F}_{\text{e2e}}$ (no matter how many parties are corrupted, $\mathcal{F}_{\text{e2e}}$ always has the "upper hand").

6.1.1 Claims Regarding $\mathcal{F}_{\text{e2e}}$

Claim 1 *Assuming the EA is not corrupted, the ideal functionality $\mathcal{F}_{\text{e2e}}^{f,Q}$ leaks no information about how honest voters vote, except for information that is revealed from the partial tally of the votes of the honest voters (according to f).*

Claim 2 *The adversary may delay the recording of an honest voter's ballot, however when it is recorded the voter obtains a receipt that enables her to verify that her vote has been properly recorded and tallied.*

Claim 3 *The receipt each voter obtains after her vote is recorded is unique; assuming the EA is honest, each receipt is independent of the voter's choice and hence can safely be passed to, e.g., a third party auditor AU.*

Claim 4 *When the EA is corrupted it is possible for the adversary to manipulate all the votes (via computational manipulation $\mathcal{M}(a_1, \ldots, a_n) = (a'_1, \ldots, a'_n)$) and even provide an incorrect tally $\hat{T}$; nevertheless, the ideal functionality ensures that honest parties are notified about whether the reported tally $\hat{T}$ and the recorded tally T' satisfy the relation Q, i.e., it returns $Q(T', \hat{T})$.*

6.1.2 Security Properties Not Captured by $\mathcal{F}_{\text{e2e}}$

We have intentionally omitted a number of security aspects from this specification of the ideal end-to-end functionality $\mathcal{F}_{\text{e2e}}$:

- **Denial of service attacks.** The ideal functionality as written enables the adversary to prevent voters from completing the ballot-casting protocol and prevent the tally from becoming available. From a definitional point of view, expressing the mitigation of such attacks is feasible by assuming certain qualities of the underlying communication and message passing mechanisms employed in the implementation. One way to extend the functionality to capture such mitigation is to oblige the adversary to deliver the $(\texttt{RecordVote})$ and $(\texttt{RecordTally})$ messages by certain deadlines. In order to do this formally, a notion of time would have to be introduced in the model, for example by introducing a global clock functionality.

- **Coercion via corrupting voters.** Even though $\mathcal{F}_{\text{e2e}}$ does not permit coercion via the receipts it provides, the adversary may still achieve coercion by corrupting a voter (e.g., hacking into the voter's computer). If this happens after the ballot-casting protocol, $\mathcal{F}_{\text{e2e}}$ reveals the choice of the voter and hence the voter may be vulnerable to coercion. Addressing this in the model is feasible by further restricting the information that is divulged when voter corruption takes place. Various intermediate levels of corruption may be considered, e.g., is the voter capable of erasing some information? rewriting some information? etc.

- **Sybil attacks.** The set of voters $V_1, \ldots, V_n$ is predetermined and integrated into the functionality $\mathcal{F}_{\text{e2e}}$. Hence, the adversary cannot manipulate the list of voters. It follows that $\mathcal{F}_{\text{e2e}}$ is applicable to the setting where the list of voters is predetermined, assumed to be public, and immune to tampering by the adversary.

Chapter 7

Architecture (Joe K./Dan) (95%)

The *architecture* of a computing system, akin to the architecture of a purely physical artifact like a bridge or a building, is its high-level structure. Just as when designing a bridge, many choices must be made when designing a computing system; these choices are driven by the system's requirements, both technical and non-functional, as well as by external factors such as the availability or affordability of computing hardware or network bandwidth. In this chapter we describe the architectural issues associated with E2EVIV systems, present a model encompassing the various possible architectural choices for such systems, and briefly explore some architectural variants.

It is important to note that we are *not* making a concrete recommendation for a specific E2EVIV system architecture. The cryptographic foundations of E2EVIV protocols have been developed to a point where we have fairly high confidence in their ability to eventually provide the required security and auditability properties. However, there are many open engineering issues associated with actually building, running, and maintaining an E2EVIV system that fulfills its requirements in the face of both routine/expected failures and a wide range of security threats. Because of these open engineering issues, architectural experimentation—preferably, empirical testing of various possible architectures to determine the most appropriate one(s) to deploy in real-world election scenarios—is vital to actually implementing a successful E2EVIV system.

7.1 Non-Functional Requirements Forcing Architectural Factors

Several non-functional requirements of E2EVIV systems force the inclusion or consideration of specific architectural factors. We consider each of these in turn.

7.1.1 Certification

Voting systems in most jurisdictions are subject to certification requirements. The EAC and NIST have developed a set of Voluntary Voting System Guidelines (VVSG) at the federal level, and the EAC launched a Voting System Testing and Certification Program in 2007; prior to 2007, voting systems were typically tested and certified by the National Association of State Election Directors (NASED). The EAC's guidelines are voluntary; however, many jurisdictions have mandated adherence to them, so an E2EVIV system that is to be widely deployed must follow them as well as any other state and local guidelines.

Certification processes generally require that a specific implementation of a system be presented to a certification laboratory; in the E2EVIV context, this would likely consist of a complete set of source files, plus binaries built to run on specific platforms and with specific external dependencies, and any custom hardware to be used in the system. Certification is typically an expensive process, often costing many times more than developing the system in the first place. Moreover, certification generally applies only to the system as submitted; any changes to the system require a full or partial recertification at additional expense.

This certification regime presents a sharp contrast to today's typical software lifecycle. Most modern software systems, especially those developed using agile techniques, are updated frequently to address implementation issues and respond to changing customer needs. For example, since its first release in September 2008 the Google Chrome web browser has seen 42 major releases—an average of about one every two months—and many minor patch releases in between. That sort of release cycle is simply not an option for an E2EVIV system, unless the voting system certification regime changes dramatically. Architectural decisions made during the system's development must therefore be undertaken with great care, because major architectural changes will be extremely expensive.

7.1.2 Abstraction

The software in an E2EVIV system must be high assurance, and must, as discussed above, undergo a certification process before it can be used in actual elections. Thus, the software must be designed and implemented with a level of abstraction that enables the generation of convincing evidence for its correctness. This requires development techniques that emphasize formal specification and verification, which divide the software system into relatively small components with well-defined, well-constrained interfaces. Each component can then be verified individually with respect to its specification; moreover, component behaviors with respect to external communication can be formally characterized in ways that allow for verification of composed subsystems.

7.1.3 Deployment

There is a wide spectrum of possible deployment scenarios for an E2EVIV system, each of which leads to certain decisions about its architecture. At one extreme, the servers for an E2EVIV system could be hosted on a bespoke server cluster, built from the ground up specifically for the system and housed in a facility under the physical control of electoral authorities or their authorized representatives. At another extreme, the servers could be hosted on a commodity cloud computing infrastructure such as Amazon EC2 or Microsoft Azure where electoral authorities have no physical control over the servers. A third extreme would see the system implemented in a purely peer-to-peer fashion, with the system's functionality distributed among all participating computers and no specifically-designated servers. The choice within this spectrum has significant impact on both system availability and system security requirements.

> 14: I'm not sure what exactly you're going for with respect to NIST component-based certification. -dmz

7.1.4 Threats

Mitigating the potential threats to E2EVIV systems also leads to various architectural choices. The following are some of the threat vectors and mitigation strategies that need to be considered.

Single Point of Failure. Any single point of failure, such as a single server that contains essential data without which the system can no longer function, is a tempting target for attack. Therefore, the architecture should attempt to minimize or eliminate such failure points.

Easy DDoS Targets. The use of fixed IP addresses (or address ranges) for E2EVIV system components can open the system to denial of service attacks, limit deployment flexibility, and make it more difficult to recover from failures. The architecture should be chosen such that IP addresses need not be hard-coded; preferably, they should be changeable be on-the-fly (i.e., during a live election scenario) if necessary to protect or restore system integrity.

DDoS Mitigation. Content distribution and network protection services such as CloudFlare [21], which route traffic for their clients through their own high-bandwidth global content distribution networks, can detect and protect against DDoS attacks. They can also provide other benefits, such as higher availability and improved response time. Such services could easily be used to deliver static data to clients in an E2EVIV system; however, although they support

technologies such as WebSockets for dynamic data, they cannot necessarily protect systems that have complex back-end architectures and client-server interactions. It is unclear whether such services can successfully be used for all interactions in E2EVIV systems without compromising privacy or other requirements; even if it is possible, doing so would certainly require specific considerations in the design of the system architecture.

Non-Standard Foundations.   One way of countering threats is to build the system using non-standard foundational techologies. For example, instead of being built atop general-purpose operating systems like Linux or NetBSD, E2EVIV system components could be implemented as *unikernels* [39], effectively single-purpose application/operating system combinations that run directly atop hypervisors such as Xen [61]. Unikernels, written using technologies such as Mirage [40] and HalVM [54], have considerably less code than general purpose operating systems; each one performs a specific task, and they communicate amongst themselves in the same manner as machines in a distributed system. This implementation style can improve security in general, by reducing the effort required to demonstrate the security of each component and by minimizing potential attack surfaces once the components are deployed.

Secure Multi-Party Computation (MPC).   A common situation in distributed computing systems is that a number of systems, each having its own secret information, want to collaboratively compute a result based on all their secret information without sharing the secret information itself. Secure multi-party computation (MPC) enables exactly this, letting the individual parties to a computation keep their local secrets while computing global functions. MPC functionality can be useful in distributing trust among multiple individuals or organizations in a complex system, mitigating a number of possible threats; we describe the distribution of trust in more detail in the following section.

7.1.5   Distributing Trust

The distribution of trust in an E2EVIV system is critical: if the system is not trustworthy, the election results generated by it are inherently suspect. The following are descriptions of the various aspects of the system that must be trusted, and some possible ways to establish that trust.

Trusting the Electoral Authority.   The public must be able to trust that the electoral authority, including elected or appointed government officials and potentially others employed or contracted by them to carry out election tasks, will not compromise the integrity of the election process or its results. In the event of a completely corrupt electoral authority, the entirety of the election process, all evidence presented to the public, and all announced results can be fabricated; thus, we need both mechanisms that minimize the possibility of a completely corrupt electoral authority and mechanisms that allow for the detection of corruption if the electoral authority includes at least one honest official.

One mechanism to help accomplish this is threshold cryptography, where the cryptographic keys that allow for results to be generated by the election system are divided into some number of shares, a smaller threshold number of which are required to actually generate results. For example, the keys might be divided into 10 shares, each of which is entrusted to a different official and at least 7 of which are required to generate the tally. This allows the election to proceed in situations where some of the election officials are unable (due to accident, illness, etc.) or unwilling (due to corruption) to produce the keys, as well as providing a check against corruption by requiring at least 7 of the 10 officials to collaborate in generating the tally. This check against corruption can be strengthened even further with public ceremonies surrounding key generation, share distribution, and tally generation, so that the specific electoral officials involved are publicly known and can be held accountable for their actions.

Another mechanism to increase trust in the electoral authority is access restriction; if the election system allows physical access only via computing systems at specific, publicly-known and well-secured locations, and only during specified time frames, it is far more difficult for a corrupt official (or group of officials) to manipulate the election results without being detected.

Finally, the use of an append-only public bulletin board to record encrypted ballot information in a tamper-evident fashion, combined with publication of sufficient information about the election protocol to allow members of the public to validate that their own ballots were cast as intended and counted as cast, also acts as a significant check against corruption by the electoral authority.

Trusting the Software. The software in an E2EVIV system must be trusted to fulfill the system's requirements. Making all development artifacts freely available as open source is an important step toward such trust, as it allows for independent verification that the source code has been designed and implemented appropriately. However, the mere fact that it is open source is not enough to make a piece of software trustworthy; there must be evidence that the running software actually corresponds to the open source code, has passed all required testing, and has not been corrupted after installation.

Several mechanisms are available for increasing the trustworthiness of deployed software. One is code signing; the system vendor can digitally sign the final binary distribution of the software and make the signature public, allowing any observer with sufficient access (in practice, this would likely be the electoral authority) to validate the signature against the actual deployed binaries. Self-certification is a variant on code signing, where the software computes a signature on itself at startup time and compares it against a known value; this can detect accidental corruption (data storage errors, spurious bit flips, etc.) but generally not malicious corruption, because an attacker that successfully corrupts any part of the software could also corrupt its self-certification mechanism.

Proof-carrying code [41] allows software to be shipped with accompanying formal correctness proofs that can be easily verified by a theorem prover or certifying compiler at runtime. Such proofs can help increase the trustworthiness of software and can easily be included in small critical sections of an E2EVIV system, but are impractical for verifying the correctness of the entire system.

Another way of increasing trust in the software is to ensure that all implemented protocols are open, with publicly available specifications, and that multiple redundant implementations of the protocols are used throughout the system. Ideally, the redundant implementations should have different low-level designs and be implemented using different programming languages. Thus, in order to corrupt the system, it would not be sufficient to corrupt a single protocol implementation; instead, all the implementations would need to be corrupted in a consistent way that could not be detected by observing differences in their behaviors.

Trusting the Servers. In addition to trust in the software, trust in the servers on which the software runs is critical. Perhaps unintuitively, one approach is to assume that the servers can't be trusted and design all the protocols and processes in the system appropriately given that assumption. The property of *software independence*—an undetected change or error in the software must not cause an undetectable change or error in an election outcome—is an example of this approach. Even if the software is fully trusted, an untrusted server can corrupt it in arbitrary ways; software independence doesn't stop such corruption, but does guarantee that it will never affect the election results, and therefore allows for recovery even though the servers cannot be trusted.

Even though distrusting the servers is useful in ensuring robust protocol and system design, steps should still be taken to make the servers more trustworthy. One way to do this is to minimize the number of components that must be trusted; an example would be using servers that support unikernel architectures, as described in Section 7.1.4, to ensure that there is little to no other potentially vulnerable software software, including OS libraries, running alongside the E2EVIV components.

Another way to increase trust in the servers is to deploy them in a dynamic, possibly public, cloud infrastructure. In this way, the particular server responsible for any given part of the system at any given time is unknown. Assuming that the entire cloud infrastructure is not compromised (for example, by a corrupt high-level insider), this can significantly reduce the likelihood of a successful attack against the servers.

Designing the system with a fully peer-to-peer architecture can help to increase its trustworthiness, as there will no longer be single points of failure; moreover, as shown in the seminal Byzantine Generals result [37], corrupted peers in a distributed system can always be detected as long as more than 2/3 of the peers remain uncorrupted. With appropriate use of cryptography, that fraction can be improved significantly such that, in the most extreme case, even a single uncorrupted peer can detect corruption of the entire rest of the peer network.

A more radical way of increasing trust in the server infrastructure as a whole is to pervasively use MPC, such that no single server ever needs to be completely trusted. While some servers may be corrupted, they will be detectable, and with a suitably redundant system design the non-corrupted servers will still be able to carry out the necessary computations for the election.

Trusting the Network.   The Internet is inherently insecure, so measures must be taken to ensure the integirity of the system's Internet-based communications. The first and most obvious of these is that all communication among components in the system should use the TLS protocol. The certificates used to establish TLS connections should be *pinned*, meaning that when a client wants to connect to a server, the client already has information about the server's security certificate (or information about a set of certificates, if more than one is acceptable). If an unexpected certificate is provided by the server, even if the hostname on the certificate matches the hostname of the server, the connection fails.

Certificates used in E2EVIV systems should have signature chains that involve not only a certificate authority (like Comodo, Symantec, etc.) but also the electoral authority or other appropriate government entity; requiring the electoral authority to be involved in the signature chain ensures that certificates cannot be issued for malicious purposes by a rogue certificate authority and used to compromise a running E2EVIV system.

Finally, all communications of critical information (ballot information, voter information, logs, audit data, etc.) in the system—even over TLS-encrypted connections—should be carried out using custom cryptographic protocols, so that even in the unlikely event of a TLS compromise the data is protected.

Trusting the Voting Client.   One clear "weak link" in the security of an E2EVIV system is the voting client, which must by definition—regardless of its implementation technology—involve untrusted machines belonging to individual voters. There are two potential issues: first, the voting client software itself might be corrupted, either in transit to a voter's machine or after installation; second, a voter's computing environment might be corrupted in a way that compromises the voter's interactions with the software.

There are various ways to ensure that the voting client software remains uncorrupted. One technique for doing so with native applications is application signing, where each application binary is digitally signed before distribution and the signature is checked at runtime. Trusted Computing techniques such as remote attestation can provide even stronger guarantees, but rely on the presence of a Trusted Platform Module (TPM) in the voter's computer. However, the fact that TPMs are not universally deployed (for example, Apple has not shipped a computer, tablet, or phone with a TPM since 2006) means that E2EVIV systems cannot rely on Trusted Computing techniques.

Implementing the voting client as a web application can alleviate some concern about the corruption of distributed native application binaries, but makes it more difficult to ensure the voting client's integrity and reason about its behavior. One problematic aspect of web applications is that JavaScript can be interpreted differently not only across platforms, but also across browsers on the same platform; there are currently four different mainstream JavaScript engines—V8 for Google Chrome, Spidermonkey for Mozilla FireFox, Nitro for Apple Safari, and Chakra for Microsoft Internet Explorer—plus a host of others used by other browsers on various platforms. In addition, the JavaScript language itself is not particularly amenable to the types of analysis required for high assurance software. Progress has been made toward enabling high assurance JavaScript, including techniques such as Certified JavaScript [36], several JavaScript tools from the Center for Advanced Software Analysis [15], and Microsoft Research's Crypto Verification Kit [23]. However, significant research is still needed in this area.

One potential approach for implementing a high assurance voting client as a web application is to start with a high-level language that support the sorts of analysis necessary to provide the required correctness and security guarantees. Mozilla's asm.js [5] is a strict subset of JavaScript that can be used as a target for compiling such higher-level languages. One nice side effect is that this subset of JavaScript is easy to reason about, and generally behaves identically across JavaScript runtime environments. With some research and tool development, it should be possible to develop a high assurance voting client, compile it to asm.js, and deploy it as a web application.

With respect to the second issue, corruption of the voter's computing environment, there is a wide range of possible threats. One, which we cannot address as part of an E2EVIV system implementation, is surreptitious logging/monitoring software that records the voter's actions and transmits them to a third party. Detection and prevention of such monitoring can be accomplished through standard secure computing practices such as malware scanning and the use of reverse firewalls (preventing the computer from making outgoing connections that are not approved by the user or by system security policies). However, we can only recommend that voters take such protective measures, and the voting client software cannot detect whether they are actually in place.

Another threat is posed by malicious web browsers or OS libraries that may deceive the voter into taking unintended actions. Such malicious code might do any or all of the following: manipulate the appearance of the voting client such that a voter believes she has voted for a particular candidate or choice when she has actually voted for a different one; block or corrupt communications between the voting client and the rest of the E2EVIV system, causing the voter to unintentionally spoil ballots or otherwise interfering with vote casting; and spoof the client-side verification screens to make a voter mistakenly believe she is voting when she is actually not interacting with the rest of E2EVIV system at all. Many of these threats can be partially or completely mitigated through judicious use cryptography and good user interface design—for example, prominent use of confirmation codes transmitted out-of-band, such as via postal mail, to authenticate communications between the voter and the system in both directions—but there will always be the possibility of malicious action on the part of untrusted client systems. Essentially, the best we can do is ensure that any individual corrupted client system cannot violate the integrity of any other part of the system.

Trusting the Data.   As previously discussed, communications can be protected using TLS and custom cryptographic protocols. However, the communicated data needs to be committed to stable storage at some point, in order to be verified, tallied, and audited. Standard database technology is inappropriate for this purpose because of the data integrity, confidentiality, and provenance requirements that must be met in an E2EVIV system. Data must be encrypted at rest to protect against straightforward physical theft of storage devices, and should remain encrypted during as much of the system's computation as possible to protect against manipulation or disclosure by either malicious software or individuals with administrative access to the data store.

One way to prevent data manipulation, or at least make it nearly infeasible, is to effectively make critical data in the system "write once" by using a technique like hash chaining to ensure its integrity; if a piece of data is changed after being written, the hash chain for all subsequently-recorded data must also be modified in order to "hide" the change. This makes data manipulation far more difficult, though still technically possible if the system is sufficiently compromised and its usage is not closely monitored (recomputing the entire hash chain would generally require significant computational resources). Replication of the data and the hash chain in multiple locations that must be accessed independently can significantly mitigate the possibility of such a compromise going undetected.

One way to prevent inappropriate disclosure of data while retaining the advantages of well-known and well-understood database technology is to use a tool like CryptDB [45], which allows for storage of and secure computation with encrypted data in standard database systems like MySQL and Postgres. Other possibilities include novel MPC frameworks like ShareMonad [38] and partial or full homomorphic cryptographic schemes such as ElGamal and BGV that enable computation of encrypted results over encrypted data.

Trusting the Voter.   The voter, like the voting client, is a "weak link" in the security of an E2EVIV system. It is possible for voters to compromise their own authentication credentials intentionally or otherwise, allowing others to vote on their behalf; to falsely challenge their own legitimately cast votes either maliciously or because of mistaken memory; and to be misled into either "voting" on a system other than the actual E2EVIV system or missing the time window for ballot casting.

In an unsupervised system, voters are responsible for their own security. However, the vast majority of people lack the knowledge and tools necessary to keep their computers secure. The electoral authority can strongly encourage certain security measures, such as use of antivirus/antimalware software on appropriate platforms, manual checking of SSL certificates to ensure that they are signed by the electoral authority, and installation of up-to-date OS, browser, and E2EVIV system software and security patches. However, there will inevitably be voters who do not follow these recommendations, and it is generally not possible—especially if it is web-based—for the E2EVIV system to evaluate a voter's compliance with them and act accordingly. Moreover, even voters using completely uncorrupted computers

can fall victim to social engineering attacks such as official-looking physical mailers with false voting codes and false voting server information.[1] Judicious selection of the distribution mechanisms for credentials and challenge mechanisms for votes may help mitigate these issues, at least to the extent that they can make it more difficult for non-malicious voters to compromise their own security.

Of course, it is impossible to defend an E2EVIV system against individual voters who are determined (or compelled) to compromise the security of their own votes; "shoulder surfing" and "rubber hose" attacks[2] remain feasible in any unsupervised voting environment, regardless of the credential distribution mechanism, voting mechanism, or other protocols implemented by the system.

Trusting the Cryptography. One area where many purportedly secure systems fall short is their cryptography; even with good intentions and sound designs, it is easy to make small but critical mistakes in cryptographic algorithm and protocol implementations that completely compromise system security.

It is clearly essential that proofs be carried out for all novel cryptographic protocols used in an E2EVIV system. These proofs may be carried out on paper, or may be mechanized such that they are checkable by computers. Moreover, all cryptographic protocol implementations must be verified against their specifications; if the specifications are mechanized, it is easier to carry out this verification, and is even possible to *synthesize* the implementation directly from the specification in a correctness-preserving fashion.

It is also essential that only well-studied secure cryptographic algorithms and pseudorandom random number generators (PRNGs) be used in E2EVIV systems. The need for secure cryptographic algorithms is obvious; the fact that bad sources of randomness can cause many vulnerabilities in the otherwise-sound cryptographic systems built atop them is less so. Exploits related directly to weak or absent random numbers have taken place in various real world systems, including fraudulent transactions in European EMV (Chip and PIN) payment systems, large-scale theft from Android-based Bitcoin wallets, and a complete compromise of the PlayStation 3 game console's digital signature process.

E2EVIV systems may use cryptographic algorithms and PRNGs approved by government agencies, such as those specified by NIST in the Federal Information Processing Standards (FIPS), but should be restricted to those that have also been extensively studied by non-government entities to counter the appearance that the government has a "back door" allowing it to defeat the system's cryptography.[3] Similarly, hardware entropy generators such as the RdRand functionality found in current Intel processors can be used to provide entropy in an E2EVIV system, but must not be the sole sources of entropy; since they are opaque subsystems whose functional details are not available for inspection, they may contain hidden weaknesses or back doors. The output of such generators can be combined with other entropy as seed input to other open-source secure PRNGs, as is standard practice in most operating systems' random number generation subsystems.

Trusting the Toolchain. Ken Thompson, in his 1984 Turing Award lecture "Reflections on Trusting Trust" [55], demonstrated a fundamental problem with trust in computing systems: an attack against the toolchain (compilers, assemblers, linkers) used to build a system can silently, and effectively undetectably, insert a "back door" or other corruption into the system. If this attack is carried out successfully, inspection of the source code for the toolchain itself and the source code for the system will show nothing unusual; the corrupted toolchain binary introduces the corruption when building itself, or when building the rest of the system, and also corrupts all the tools that can be used to analyze the system (disassemblers, binary dump tools, etc.) such that the corruption remains hidden.

---

[1] Social engineering attacks have often been used in non-Internet election scenarios, primarily with the goal of suppressing votes; durng the 2014 midterm election cycle, there were numerous reports of mailings directing voters to incorrect voting places, giving incorrect election dates and absentee ballot submission deadlines, and containing incorrect information about voter registration.

[2] These are both actual terms of art in the field of computer security. A "shoulder surfing" adversary monitors all the actions of the voter throughout the voting process, while an adversary employing a "rubber hose" attack extracts information from the voter or forces the voter to take particular actions by means of threatened or actual physical harm.

[3] As reported by the New York Times in late 2013, the NSA actually did insert a back door into the NIST-approved Dual_EC_DRBG PRNG (it has since been removed from the approved PRNG list) and has actively worked to insert similar back doors into other cryptographic systems.

Thompson himself successfully carried out such an attack within Bell Labs, and similar attacks have occurred "in the wild" against systems such as the Delphi development environment for Windows application; with stakes as high as controlling national election results, it is not a stretch to believe that such attacks would be attempted against E2EVIV systems.

There are multiple ways to mitigate the possible impact of such an attack. One is to ensure that the system uses a diverse set of implementations of key components, all based on the same specification but with different source code, built with different compilers, and preferably running on different hardware and OS platforms; corruption of a single component, or even a small number of them, could then be detected by the uncorrupted components, and the effort required to corrupt the system as a whole would be much higher. Another is to counter the possibility of Thompson-style exploits by using multiple toolchains in the technique proposed by David A. Wheeler in his Ph.D. thesis, "Fully Countering Trusting Trust through Diverse Double-Compiling" [60].

7.1.6 Scalability

An E2EVIV system, particularly at the national level, must be able to handle a wide range of demand. It is human nature that many voters will wait until the last day, or even the last hour, of a voting period to cast their votes. Moreover, it is likely that attacks against the system—and thus, system activity in general—will increase in intensity as the end of a voting period approaches. Thus, while the system may see very little sustained activity for much of an election period, it must be able to scale to extreme levels of activity at peak times. The architecture must take this into account, so that the system can be dynamically deployed on more computing and network resources as need arises. This might be done either by utilizing public cloud resources that support elastic demand or by using private resources that can be brought on- and offline as required.

7.1.7 Availability

E2EVIV systems must exhibit high availability; Chapter 5 stated an explicit requirement for 99.9% uptime during election periods and the ability to recover from generalized (i.e., not caused by natural disaster or malicious attack on the system) failures in under 10 minutes, and higher availability—including in the face of malicious attack—would be preferable. There are a number of techniques for ensuring high availablity of systems, including the use of services like those provided by Cloudflare to handle traffic spikes and distributed denial of service attacks. The system architecture should be constructed in a way that does not foreclose the use of such techniques.

7.1.8 Usability

Usability, including accessibility for disabled voters, is of paramount importance in an E2EVIV system. Especially for the voter-facing parts of the system, the choice of implementation technology may have a significant effect on usability. Essentially, choices may need to be made between using Web technologies, which have significant advantages in terms of reach (cross-platform, able to be used on various sizes of device), and native applications, which tend to exhibit richer interaction design and support more accessibility features. The architecture might also allow for both types of implementation, potentially at the cost of additional architectural complexity.

7.2 Architectural Feature Model

As we have seen, there are many considerations to take into account when making architectural decisions about an E2EVIV system. Here, we model the various architectural dimensions, and the (possibly wide) range of choices within each, to give a sense of the potential solution space for a workable E2EVIV system architecture.

The Business Object Notation diagram in Figure 7.1 shows the architectural choices that need to be made; for each attribute of the architecture, a list of possible choices is provided. There are seven dimensions, each of which can take on a set of values. The values are chosen from 2-element sets for four of the dimensions, from a 3-element for one, and from 4-element sets for the remaining two. Since each dimension must have at least one selection (the empty set is disallowed), this yields a total of $(2^2 - 1)^4 \times (2^3 - 1) \times (2^4 - 1)^2 = 127{,}575$ possible architectural variants.

The architectural dimensions we have identified are the following:

- **Distribution of Authority** Authority in the system—that is, the "official" set of data stored in the system and control over access to and manipulation of that data—can be centralized or distributed. Centralized authority eliminates concerns about data consistency, as data can only be manipulated by one entity, but may cause issues related to system responsiveness, availability, and reliability. Distributed authority eliminates a single point of failure at the expense of needing to ensure data consistency and integrity. It is also possible to implement a hybrid authority model, where authority is concentrated in a small set of entities relative to the system as a whole; this is technically distributed authority because it is spread across entities, but behaves like centralized authority from the perspective of most of the entities in the system.

- **Cryptographic Protocols** The set of cryptographic algorithms and protocols used to protect voter privacy and insure ballot integrity is a critical component in any E2EVIV system. Security characterizations of individual cryptographic algorithms (e.g., block ciphers such as AES, standard public key cryptosystems such as RSA, threshold cryptosystems such as ElGamal) can be found in the large body of cryptography literature, and the selection of individual algorithms is generally a matter of picking an appropriate algorithm and security strength for a given task. However, since novel cryptographic protocols such as those required to implement E2EVIV systems are not widely used or studied, evidence must be provided for the security of any such protocols used in a deployed system. Such evidence can be provided in two basic ways: through "paper" proofs that are carefully checked by multiple experts, or, if the protocols are mechanized in a formal specification language, through proofs that are automatically generated by cryptography protocol verifiers such as ProVerif [9]. In general, it would be preferable for all cryptographic protocols in the system to be mechanized, as evidence could be generated repeatably and easily regenerated in the event of minor protocol changes; however, there may be cases where this is impractical. Thus, the cryptographic protocols of the system may have "paper" specifications, be mechanized in a formal system, or some combination thereof. Moreover, their implementations may be verified against the specifications in various ways, or may be directly synthesized from the specifications in a way that guarantees correctness.

- **Evidence of Correctness** The development of a high assurance system such as an E2EVIV system proceeds from a specification, at some level of formality, to an implementation that is intended to fulfill the specification. Assurance that the implementation actually does fulfill the specification can generally be obtained in two ways. First, the implementation can be developed in a way such that it is mechanically tied to the specification; for example, code generation techniques and refinement techniques can be used to mechanically generate correct implementations from specifications. Second, the implementation can be developed "by hand" with a set of included assertions that are meant to establish that the specification is being faithfully implemented; these assertions may then be checked by code analysis tools when building the system, or may be automatically compiled into testing code that validates the assertions when running the system. In practice, while it is clearly desirable for as much of the implementation as possible to be mechanically generated from the specification, most high assurance systems use some combination of these two techniques.

- **Implementation Type** Regardless of the choices made along any of the other dimensions, a reference for what constitutes a "correct" implementation must be provided. Either a "golden implementation" with strong correctness guarantees or a complete set of open protocols and specifications may serve as such a reference.

  A golden implementation G may be synthesized directly from a formal specification using semantics-preserving transformations, may be implemented in one of several languages using a "correct by construction" approach, or may be painstakingly verified against a formal specification either by hand or in an automated fashion. In any case, such a G defines the correct behavior of the system but may—because of its implementation technology

```
static_diagram E2EVIV_Architecture_Dimensions
  -- This diagram shows the various dimensions of an E2EVIV architecture
component
  class E2EVIV_ARCHITECTURE
  feature
    authority_distribution: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { Centralized, Distributed };
      end
    crypto_protocols: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { On_Paper, Mechanized,
                                      Verified, Generated };
      end
    correctness_evidence: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { Process_Based, Assertions };
      end
    implementation_type: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { Golden_Implementation,
                                      Open_Protocols_and_Specs };
      end
    key_distribution_method: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { Public_Ceremony, Threshold_Cryptography,
                                      PKI, Web_of_Trust };
      end
    deployment_style: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { Trusted_Servers, Public_Cloud, Peer_to_Peer };
      end
    client_technology: SET[VALUE]
      ensure 0 < Result.count;
             for_all v: VALUE such_that v member_of Result
               it_holds v member_of { Custom_App, Web_Based };
      end
  end
end
```

Figure 7.1: A specification of the possible variants for an E2EVIV system.

or other factors—not satisfy real-world performance requirements, memory usage/storage requirements, etc. However, G can be used as a reference point for other implementations; if the behavior of an implementation X, which does satisfy deployment requirements but for which we have no strong correctness guarantees, conforms to that of G, then X is also a correct implementation (and is likely better suited for deployment).

In the absence of a golden implementation, evidence for implementation correctness can come in the form of conformance testing against open protocols and specifications. This requires that such protocols and specifications are provided for the entire system in formats such that conformance checking is feasible.

- **Key Distribution Method** In all the possible implementations of an E2EVIV system, encryption keys will be required to fulfill security, privacy, and integrity requirements. The ways in which these keys will be determined by the cryptographic algorithms chosen for use in the system, but the ways in which they are distributed depend on architectural choices. For example, public "key generation and distribution" ceremonies and the use of threshold cryptography for the keys that enable generation of the final tally, described in Section 7.1.5, may be used to improve trust in the system and making it more resilient in the face of partial corruption of the electoral authority. However, the generation and use of encryption keys in the system is not limited to the election authority; as a result of the pervasive use of TLS encryption for communication, keys are continually generated and exchanged throughout the system.

  The most widespread method of key distribution today is the public key infrastructure (PKI), which manages digital certificates and related artifacts. Trusted certificate authorities issue certificates that bind public keys with user or organizational identities; this is what allows a user to connect to her bank online, examine the certificate presented during the connection, and see that the certificate was in fact issued to her bank rather than a man-in-the-middle attacker. As described in Section 7.1.5, PKI can be used to guarantee to voters (in the event that they check) that their connections to an E2EVIV system are secured using a certificate issued to and signed by the electoral authority. PKI is effectively mandatory for secure Internet connections using the TLS protocol, but may also be used to manage other keys in the system; however, this requires a trusted certificate authority for such keys.

  Another method of key distribution is the "web of trust", first described by the creator of Pretty Good Privacy (PGP) [65] in 1992. Unlike a typical PKI system, a web of trust system does not have the concept of trusted certificate authorities; instead, it is essentially a large collection of public keys that can be digitally signed as a signal of trust. Each public key in the repository identifies an entity (individual or organization) that possesses the corresponding private key. Each public key can be digitally signed by the owners of other keys. For example, assume Alice and Bob have both published their public keys, which are A and B respectively. If Alice knows Bob personally, and Bob can demonstrate to Alice that his public key is actually B, then Alice can sign key B (with her private key) to indicate her trust that key B belongs to Bob. Such signatures build a distributed set of trust relationships: if Alice trusts Bob, and Bob trusts Charlie, then it is presumed that Alice can trust Charlie. This is similar to the way certificate authorities work in PKI systems; in effect, Bob acts as a certificate authority for Alice because of the trust relationship they have established, and Alice can trust any key that he has signed (or that has been signed by a key he has signed, etc.). Key distribution via a web of trust could be useful in various parts of E2EVIV systems, as a means of collectively building trust relationships within the system rather than relying on the single points of failure inherent in PKI infrastructures.

- **Deployment Style** The E2EVIV software can be deployed in one of three ways: (1) servers run entirely on trusted servers managed by the electoral authority or its designated representatives, and client applications access the servers through well-defined, well-controlled interfaces; (2) servers run, in whole or in part, on public cloud infrastructure, while client applications still access them through well-defined interfaces; (3) the system is structured in a peer-to-peer fashion, where "server" functionality is distributed across all entities in the system and at least some of them run in an uncontrolled environment (i.e., voters' computers).

- **Client Technology** Implementation of the client software used by voters, as well as the administration software used by the electoral authority, can be done in two basic ways: (1) develop custom applications for the various hardware/OS platforms that will be used by voters and the electoral authority; or (2) use Web application technologies to develop a single Web-based application that will be accessible from all (reasonable) platforms. It
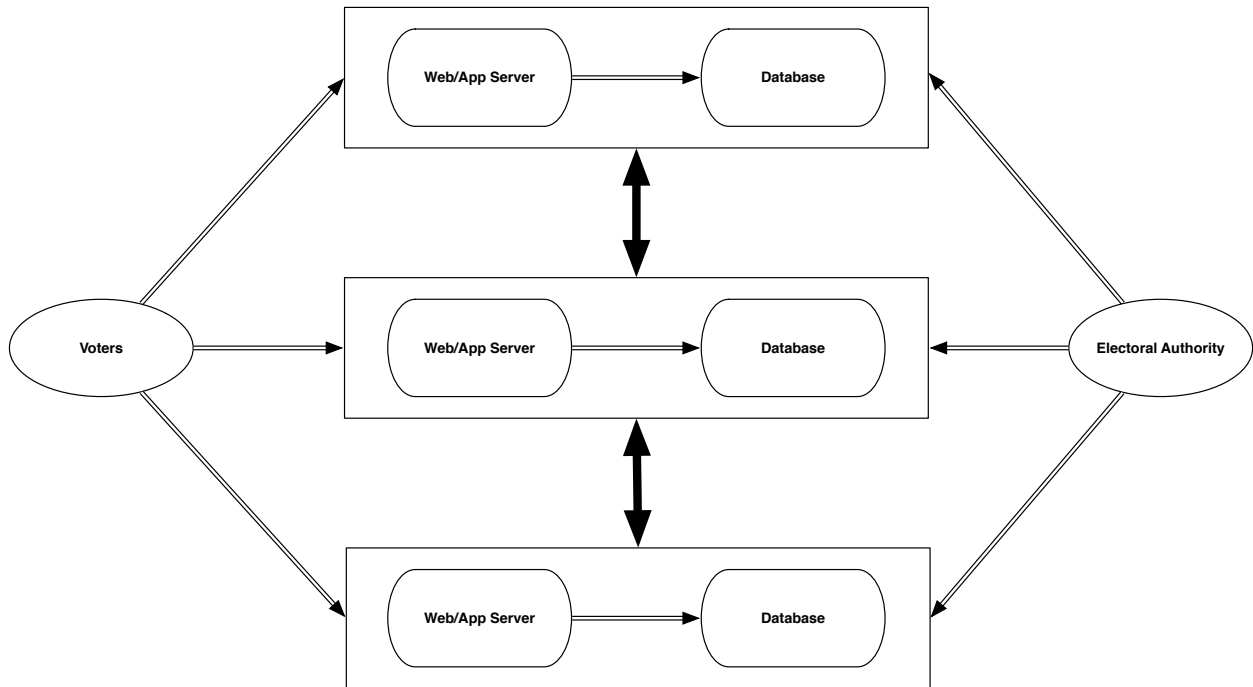
Figure 7.2: An architecture with centralized authority and mirrored servers.

is also possible to choose both implementation strategies for all applications (i.e., both voters and the electoral authority can access the system through either native applications or a Web application, as they choose) or make different choices for different applications (e.g., the voter application is implemented as a Web application while the electoral authority's administration application is implemented as a native application).

7.3   Primary Architectural Variants

Given the many dimensions of the architectural feature model, and number of choices in each, the number of possible architectural variants for the system is large. Here, we briefly discuss a few of the primary system variants that can be described by the feature model. Since we are only describing them at a high level, some of the variants correspond to many possible feature selections in the feature model (for example, they might have any of the different types of correctness evidence or any of the different key distribution methods).

7.3.1   Mirrored Servers

One possible architecture, which features centralized authority as its primary defining characteristic, is the "mirrored servers" architecture depicted in Figure 7.2. The double arrows in this diagram (and later diagrams in this chapter) denote *client* relationships (one entity making use of another's services), while the thick double-ended arrows denote *mirroring* relationships (entities, or groups of entities, ensuring that their states accurately reflect each other for redundancy or availability).

In this architecture the Web/App Server (to which voters, using either a web-based interface or custom applications, connect to cast their ballots) is a client of the Database, which stores all information relevant to the operation of the E2EVIV system (ballot styles, cast and spoiled ballots, etc.). In an actual implementation, the monolithic Database would likely be split into multiple databases since the access patterns and performance needs for data such as ballot styles, cast and spoiled ballots, voter lists, etc., are likely to be quite different. There might also be more components within each mirror (for example, separate servers for dealing with native applications vs. web access in a system that supports both).

Regardless of the number of servers within each mirror, the mirroring in this architecture is done primarily for availability and reliability; it ensures that, as long as at least one set of mirrored servers is running, the system can remain operational (albeit perhaps at a degraded level of responsiveness). Authority is centralized in the sense that each mirror has a complete set of data for the system and behaves accordingly; one mirror is designated as the *primary* mirror and is considered the authoritative source of information in the event of inconsistency. Voters and the electoral authority access the system by interacting with an individual (typically, the primary) mirror, and the entire set of mirrors appears logically as a single server-side system.

7.3.2   Large Fixed Set of Servers

Another possible architecture, which introduces the potential for distributed authority but still has the logical presentation of a single server-side system, is a large fixed set of servers. This architecture, an example of which is depicted in Figure 7.3, still features mirroring for redundancy and availability; however, it allows for flexible allocation of resources. For example, there might be twice as many Web/App Server instances as there are Database instances, or there might be more Database instances dealing with dynamic cast and spoiled ballot data than dealing with fixed election definition data such as ballot styles.

The servers within this architecture could, amongst themselves, behave as a peer-to-peer system, a set of client-server systems, or a set of mirrors of various sizes for the purposes of providing high availability and redundant storage and ensuring data consistency. A key aspect of this architecture is that the number of servers, while large, is fixed; this allows the topology of the servers and the communications amongst them to be known at all times, making it straightforward to monitor the system's health and performance and to quickly detect any issues that arise.

As an example, Figure 7.3—only one of many possible server topologies in such an architecture—has two separate mirrored Databases (one with two mirrors, and one with three) being accessed by three separate Web/App Servers. If it is determined that the Databases are underloaded and the Web/App Servers are overloaded, one of the servers running Database B could easily be repurposed to run an additional Web/App Server (Figure 7.4) without changing the actual set of servers in the architecture and without compromising the redundancy of data storage in the system.

While one possible deployment of this architecture would see every server containing the full authoritative data set, it is far more likely that each would contain only part of it and that the authority in the system would, therefore, follow either a hybrid or a distributed model.

7.3.3   Dynamic Cloud

The two previous architectural variants involved the deployment of a fixed set of servers, either as a collection of mirrors or in other topologies. The next variant departs from these by deploying services not across a fixed set of servers, but instead within a dynamic cloud infrastructure, while still presenting itself as a single server-side system for external interactions. Such an infrastructure allows for the addition and removal of computing resources as necessary during the operation of the system, using various distributed communication and consistency protocols to deal with resource changes in a way that is effectively invisible to the system's users while maintaining data integrity and service availability. Figures 7.5 and 7.6 show snapshots of a dynamic cloud deployment at times when it has five and eleven running servers, respectively. Note, in particular, that the client relationships among the servers in the cloud may evolve over time as well; for example, in Figure 7.5, the server at the "top" of the cloud could establish direct communication with the server at the "bottom left" of the cloud if necessary.
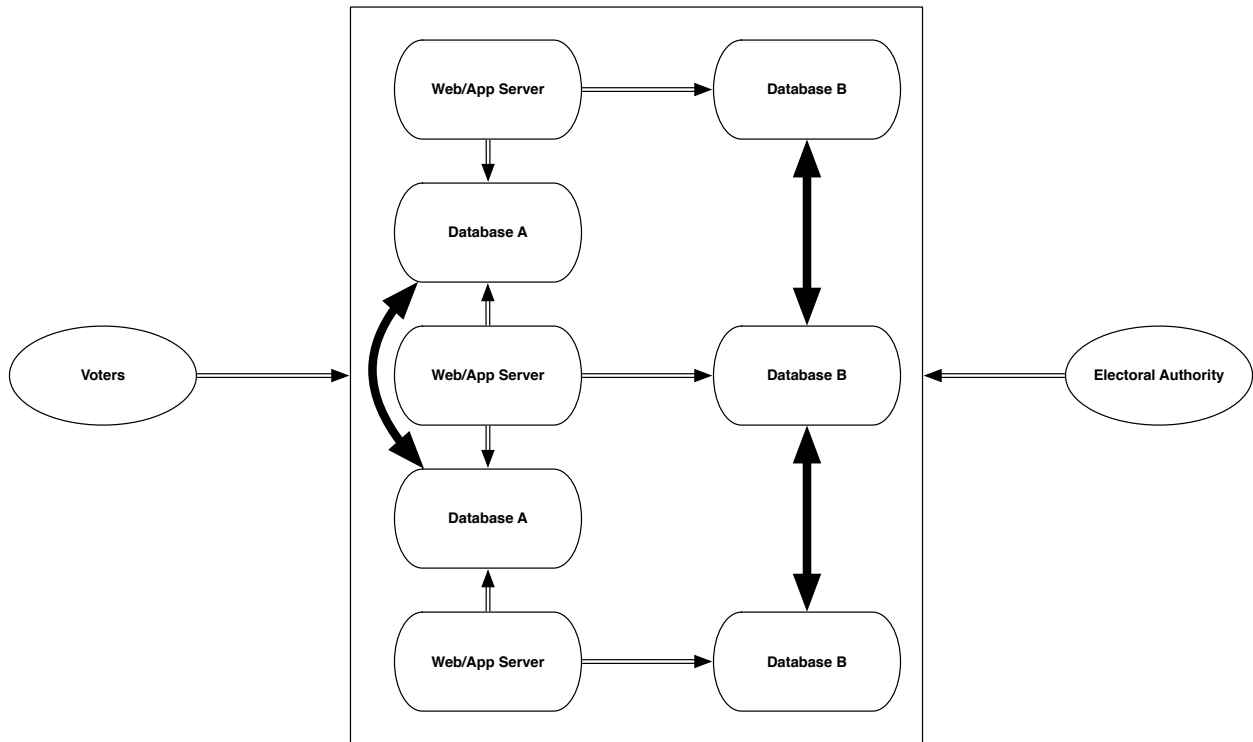
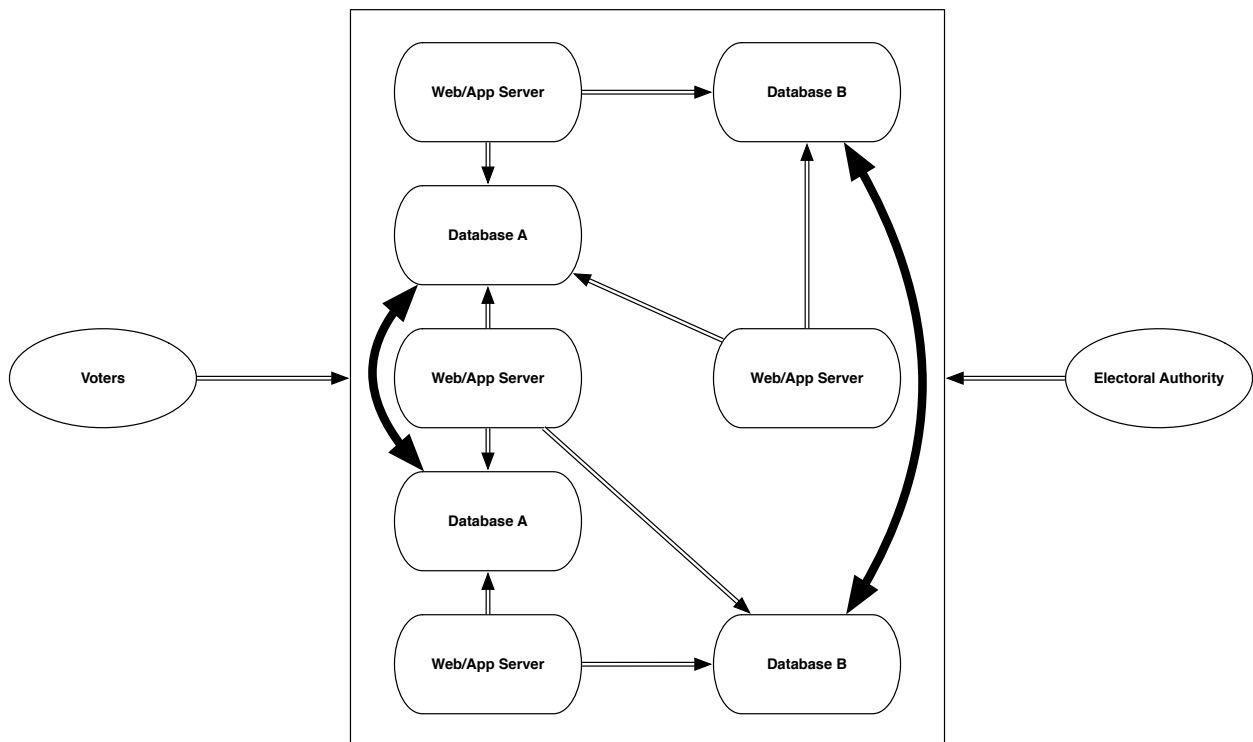Figure 7.3: An architecture with a large fixed set of servers.

Figure 7.4: The same fixed set of servers as in Figure 7.3 performing a different allocation of tasks.
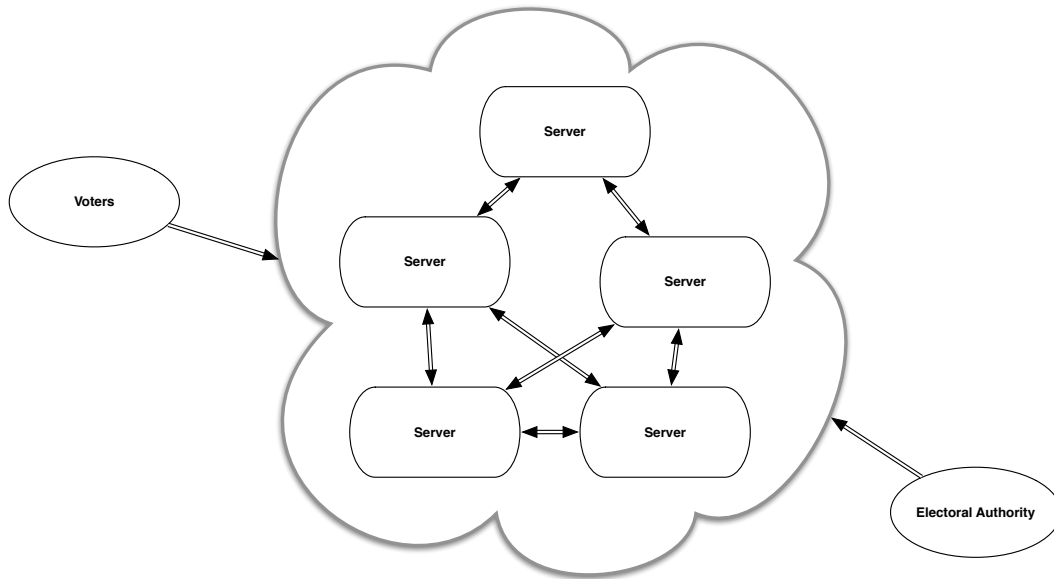
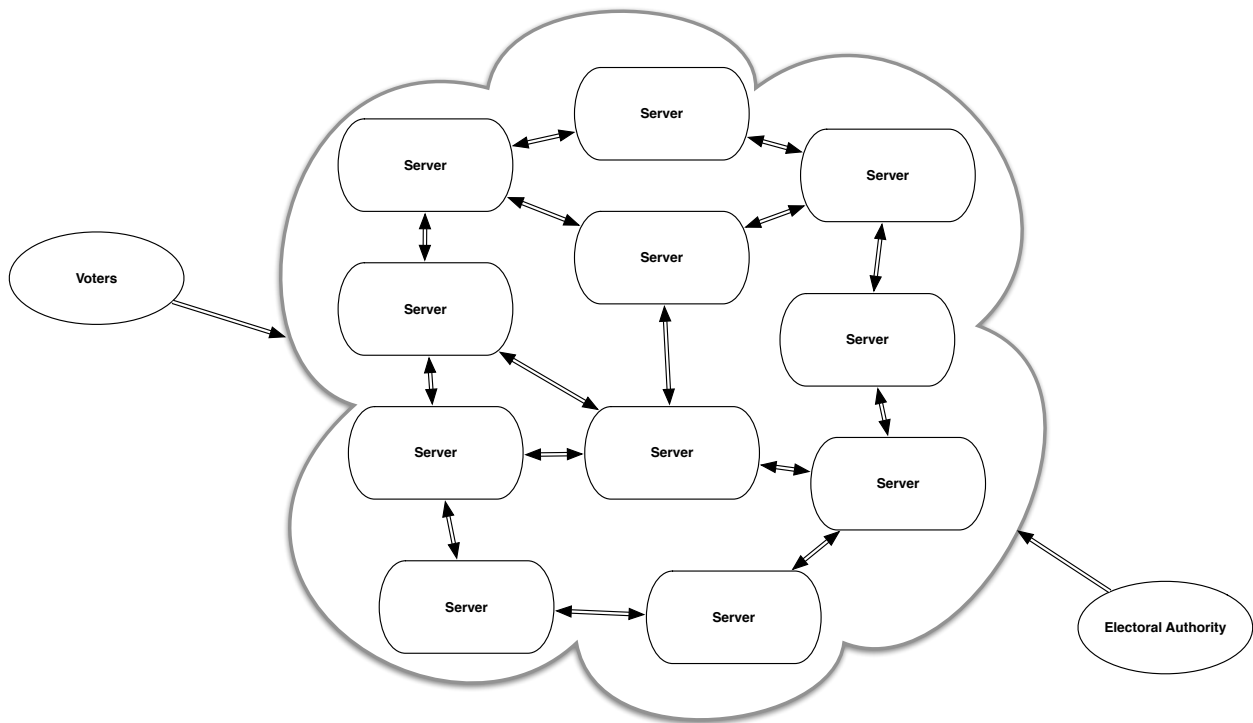Figure 7.5: A dynamic cloud architecture with a small number of servers.

Figure 7.6: A dynamic cloud architecture with a larger number of servers.

Effectively, a dynamic cloud deployment behaves similarly to a deployment with a large fixed number of servers; the main difference is that the number of servers is variable. This allows for the system to initially consume minimal resources, expanding or contracting as necessary (within the bounds of the dynamic cloud) to maintain acceptable response time and availability in the face of elastic demand.

Despite the use of the word "cloud", a dynamic cloud architecture need not actually be deployed on a public cloud infrastructure; private cloud infrastructures consisting of only trusted servers may be built as necessary to support the system.[4] Regardless of whether the system is deployed on a trusted or public infrastructure, authority in a dynamic cloud architecture follows either a fully distributed or a hybrid model; some servers in the cloud may have authority over others, or they may interact using consensus protocols or similar mechanisms.

7.3.4 Peer-to-Peer

In all the architectural variants described so far, the system presents itself as a single "server" regardless of its "internal" network topology. In a *peer-to-peer* implementation, the computational work of the system is distributed across all the participants and there is no clearly defined distinction between "client" and "server". For example, Figure 7.7 depicts a peer-to-peer system with a number of peers belonging to individual voters, some belonging to political parties (A, B and C), and some belonging to the electoral authority. The double-headed arrows in the figure represent communication links among the peers; for example, if the upper-left peer belonging to Party A needs to communicate with the lower-right peer belonging to the electoral authority, it must send a message that travels across at least 7 communication links. The communication links in a peer-to-peer network typically change over time, based on each peer's knowledge about its network environment and the locations of other peers.

Authority in a peer-to-peer architecture is fully distributed. In the case of an E2EVIV system, the electoral authority would set up and maintain some trusted peers as a way to "bootstrap" the peer-to-peer network, and political organizations (parties, lobbying groups, etc.) might also choose to maintain peers, perhaps with their own implementations of the election software in a system designed with open protocols and specifications, as a way of participating in the electoral process and strengthening trust in the results. Individual voters running the software on their own machines would also be peers for the duration of their voting sessions (or longer, if they chose to contribute to the management of the election by leaving the software running); effectively, a peer-to-peer architecture is a way of "crowdsourcing" the resources required to run election system.

A peer-to-peer architecture raises significant security concerns that differ from those of the other architectures we have described. While some of the computer systems controlled by the electoral authority might be trusted, the vast majority of systems belonging to individual voters or political organzations will certainly not be; it is impossible to run a peer-to-peer E2EVIV system using only trusted computing resources.[5] It is therefore important to ensure that no corrupt peer, or set of corrupt peers, can undetectably compromise election results, violate voter privacy, or otherwise violate the E2EVIV system requirements.

One way to address this problem is to employ a *blockchain*, like that used in Bitcoin and other cryptocurrencies, to log critical election information (cast and spoiled ballots, the fact that a given voter has voted in the election, etc.). A blockchain is a public write-only ledger, collectively maintained by the peers in the system, that records a sequence of events. The mechanism by which this recording is done ensures that the peers reach a consensus about the events that have occurred and their ordering, and that once an event (such as the casting of an encrypted ballot) has been placed in the ledger it can be neither modified nor reordered with respect to other events. As long as more than half of the peer computing power in the network is "honest" and follows the correct protocol, the integrity of the ledger is guaranteed. At any given time, it is likely that the computing power contributed by the electoral authority and high-profile political organizations—which can be hosted on trusted, closely-monitored computing systems—will vastly outweigh the computing power contributed by individual voters during their ballot casting sessions; moreover,

---

[4]In the E2EVIV context, however, public cloud infrastructures are likely preferable for economic reasons; it is virtually inconceivable that an electoral authority or its suppliers could build a cloud infrastructure with scale and reliability comparable to existing public cloud infrastructures at reasonable cost.

[5]This is true of "pure" peer-to-peer architectures of the type we are discussing in this section; an architecture where only the *servers* interact in a peer-to-peer fashion while presenting a single interface or set of interfaces to clients can, as previously noted, be implemented on a fixed set of servers or in a dynamic cloud.
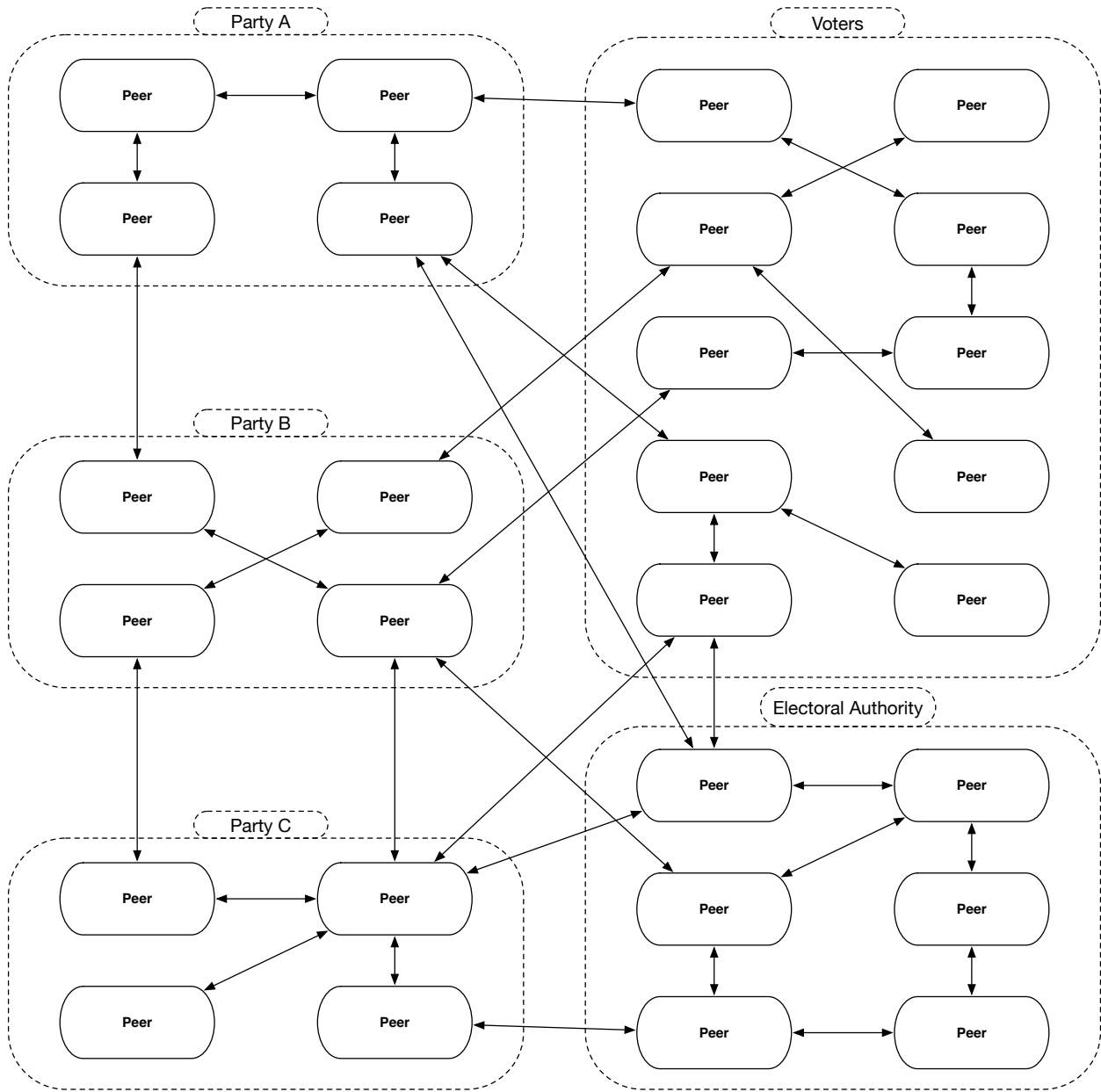
Figure 7.7: A peer-to-peer architecture.

the situation where more than half the peer computing power is dishonest can be detected (by the honest part of the network, or by external observers) and dealt with in various ways. Thus, maintaining the integrity of a blockchain should be reasonably straightforward in an E2EVIV system. However, other aspects of implementing a peer-to-peer architecture—such as distribution of the computing client to voters and organizations, achieving sufficient ease of use and performance, etc.—may prove more difficult.

7.4 Summary

As can be seen from the many architectural dimensions we have described and the primary architectural variants we have briefly discussed, there are many different ways in which an E2EVIV system could be designed and implemented. It is not clear which of the primary variants would be the "best" option, nor is it clear exactly what criteria would be used to make that determination among multiple architectures that fulfill all the E2EVIV requirements. Further research and experimentation is therefore necessary to determine a suitable path forward for E2EVIV implementation and deployment.

Chapter 8

Rigorous Software Engineering (Joe K./Dan/Adam) (40%)

Sound engineering practices are the foundation for building reliable and secure software of any type. This is particularly true for critical systems which must be trusted to perform important tasks correctly, and where the consequences for failure threaten lives, political system integrity, and property.

This section introduces methodologies that help reduce errors and improve confidence in software development. We avoid discussion of particular technologies except to illustrate our recommended methodologies with examples in pracice.

8.1 Informal System Specifications

8.1.1 Domain Models

8.1.2 Requirements and Scenarios

15: traceability and automation; reflect back upon requirements and scenarios previously described

8.1.3 Concept Specifications

8.1.4 Utility of Informal Specifications

8.2 Formal System Specifications

8.2.1 Architecture Specifications

8.2.2 Concept Specifications

8.2.3 Source Code Specifications

8.2.4 Protocol Specifications

8.2.5 Utility of Formal Specifications

8.3 Methodology

8.3.1 Engineering Methodology

> 16: Introduce context with NIST, ISO standards for methodology?

> 17: Lay out the characteristics we want more explicitly, eg, freedom from flaws, repeatability, correctness

Version Control

Version control systems (VCS) manage changes between the versions of a project as it evolves during the course of development. Revision control is the preferred way to share software artifacts across a team, but all software projects, even those developed by teams as small as one person should use VCS.

In general, a developer uses the VCS first to "check out" the files comprising a project into her "working copy". Then, after making changes to those files, the developer "checks in" or "commits" the changes to the VCS. After committing, those changes are available for other developers to integrate into their working copies.

When different sets of changes have been made by multiple developers, the VCS can merge those changes either automatically or by using developer input to ensure the project remains consistent. The ability to merge changes is critical for teams of developers who work concurrently on a single project, and is the reason that file sharing tools like Dropbox or Google Drive are an inadequate substitute for a VCS.

VCS is particularly important for projects that must be audited. An entry to a project log is created every time a developer commits their work. Any file in the project can be inspected to show its provenance, even down to the level of which line was committed by which developer on what date. Some VCS tools also optionally allow for commits to be cryptographically signed, offering assurance that, for example, the changes have been audited by a trusted authority before being integrated into the project [16].

Moving from simple file storage to VCS is a tremendous improvement for development, but poor use of VCS can negate many of the potential benefits. For example, the log built from the commits of developers is of less use to auditors if the changes in each commit are not clearly associated with a particular new feature or bug fix. Likewise if developers commit changes in a broken state, other developers' productivity suffers and it becomes more difficult later to isolate which commit introduced a bug. Each VCS supports multiple workflow practices that should be adopted in order to limit these problems and get the most benefit from VCS use [7][44].

Issue Tracking

During much of the software development and maintenance process, teams add features and fix bugs. In an issue tracking system, each new feature, each reported bug, and other discrete development tasks are tracked as issues from creation to implementation, review, testing, and integration. Issues can be organized by metadata such as assignee, project milestone, priority, and task type. Issue trackers are essentially to-do lists with additional structure that is specialized to support effective software engineering.

These issues and their metadata give team members a view into the status and health of the project, and facilitates the implementation of project management methodologies . For example, the issue tracker may automatically require a code review step before an issue can be resolved. Issue trackers help teams make fewer mistakes when following best-practice software engineering workflows.

18: reference management section

Team members can annotate issues with comments or attach supplementary documents, creating a record of design decisions and thought processes. This information is invaluable when investigating future bugs or making subsequent changes to a design, and is often lost when such discussions take place out-of-band and lose their association with the task that motivated them.

Most issue trackers integrate with VCS in order to associate issues with the source code changes that were made in order to complete those tasks. When combined with the design discussions captured in issue comments and attachments, this further enhances the ability of the team to reliably understand and maintain the project in the future.

Issue trackers not only benefit the project team, but also often serve as a first line of contact for users of a system when they encounter problems. In projects with short timelines like election systems, it is critical to incorporate feedback into development as quickly as possible. Giving users or front-line support staff the power to create issues directly makes the feedback loop very small.

Keeping a public issue tracker reduces duplicated effort across both users and developers. If a system has a flaw, that flaw will likely become apparent to multiple users, and duplicate reports are less likely if users can check the issue tracker for other reports of similar problems. The development team can then coordinate an effective response to problems. The team can triage issues by importance and urgency, discuss potential solutions, assign developers to implement those solutions, and finally make sure the problem is resolved and notify the users who originally reported the problem.

Testing

Software testing practices are a key component of any software engineering methodology. Even when parts of a system are formally verified, testing helps provide additional assurance that the system:

- satisfies its requirements,
- behaves as expected for particular inputs,
- works correctly in diverse environments,
- and has sufficient performance.

Testing serves the key functions of uncovering flaws quickly, and ensuring that previously-fixed flaws do not recur in later versions. The assurance testing provides comes from performing these functions. It is impossible for testing to reveal all possible flaws in any realistic program – the number of possible inputs for any such system is so large that an exhaustive test would effectively run forever – but tests that successfully reveal and prevent recurrences of some flaws provide more assurance than an untested system.

Formal verification and testing should not be viewed as opposing alternatives, but rather as complementary techniques that together provide assurance in the developed software. It is not feasible to exhaustively test every possible input and every possible path through any non-trivial program, while formal techniques offer guarantees over all possible inputs. However, formal techniques usually cannot scale to provide those guarantees for entire systems, and sometimes must make simplifying assumptions about the environment in which software runs, or reason about a simplified model of the actual system. Since testing can use the real system in a real environment, it can uncover flaws that are beyond the scope or capability of the formal techniques.

Different testing practices provide complementary types of assurance; no single testing practice is sufficient to cover the above list. Instead, multiple types of testing should be used in combination on every project.

Unit Testing   Unit testing exercises the smallest components (the "units") of a program that are feasible to test. The granularity of unit tests varies by programming language, but typically unit tests are small enough to test the implementation within a single module, class, or other per-file abstraction.

Unit tests are usually written to reflect the specification of the unit under test. For example, a unit that implements a specification of addition might have unit tests that check the associativity and commutativity of the implementation. Most software specifications are too abstract to translate directly into unit tests, so for a property like associativity, developers must choose particular concrete values to test. The shortcoming here is that developers might not choose the particular values that would expose a flaw, and so the test suite will succeed despite the presence of that flaw.

Developer intuition and understanding of the implementation increases the likelihood that unit tests will exercise code containing a flaw, but tests still cannot be exhaustive. Code coverage, the percentage of lines of code exercised by a given test suite, is often used to measure the effectiveness of unit tests. In practice, high coverage percentages have not been shown to necessarily uncover more flaws [35], however more sophisticated coverage measures such as branch coverage are more promising [30].

Randomized Testing   Manually-written tests can only exercise a small fraction of the potential inputs to a program. When test cases can be generated randomly, it is much cheaper to produce a large number of test cases that can explore a larger fraction of potential inputs.

With randomized testing, developers specify a means for generating the inputs to a test, and provide a function or "oracle" for evaluating whether the test succeeded with those inputs. These test generators can more closely mirror specifications than tests that use concrete values, as they can instead make assertions about all possible values.

For example, a unit test for an addition implementation might by hand assert that $0+0 = 0$, $1+0 = 0$, and $2^{32}+0 = 0$, but a random test could assert that for all integers x, $x+0 = 0$. Unless the range of the input is very small, a random test will not provide an exhaustive proof of the property it expresses. It can, however, easily provide orders of magnitude more test cases than hand-written tests, and will usually produce test cases that a developer might not think to add intuitively. When a higher level of assurance is required, the specification of a random test often translates directly to a logical formula usable by formal techniques, easing the transition to a verified system [53].

In the addition example above, it is straightforward to generate random integers and check whether the result is zero. However, randomized testing is situational, since an oracle is not always straightforward to develop, and even the random input generator can be quite complicated for complex input types. For example, random testing has successfully been used to find flaws in C compilers. The development of the C test program generator has itself been the subject of extensive research [62], and the oracles used are primarily other C compilers.

Because it is difficult to apply random testing to programs with complex input types and no readily-available oracles, random testing is most commonly applied to unit tests. Random testing can still be useful in such cases by relaxing the requirements of the input generator and oracle, and simply observing whether a program crashes or violates internal assertions when presented with a random, possibly malformed input. This variant of random testing is known as "fuzzing", and should be used as a complement to other forms of testing.

Model-Based Testing Another method for automatically generating test cases for a program is model-based testing, where formal models of the program's behavior can be used as test oracles, as guides for choosing test data, or both.

Formal models of program behavior can be used as test oracles by compiling them into the software as *runtime assertion checks*. For example, assume the specification for some function f guarantees that some condition x is true when it returns. If x is ever false when f returns, f does not satisfy its specification. Any good set of unit tests for f should detect this divergence, and it would certainly be straightforward to write (by hand) a test oracle that checked the value of x after executing f. However, in many cases f's specification can be *automatically* transformed into a set of runtime assertions, such that (in this case) an error is always raised if x is false when f returns. Compilers that perform such transformations are available for several programming languages and corresponding specification languages; some widely-used examples are Java and the Java Modeling Language (JML), C# and Code Contracts, C and the Executable ANSI/ISO C Specification Language (E-ACSL), and Eiffel (which has an integrated specification language).

The set of runtime assertion checks derived from a formal specification of a module effectively become a test oracle for that module; in general, more detailed specifications lead to better test oracles. For a function f with such an oracle, each test is run by simply calling f with some input. It is possible that the specification of f restricts the set of possible inputs; if so, tests that supply invalid inputs are effectively meaningless and their output is ignored. A test that supplies valid inputs passes if all the runtime assertion checks pass and fails if any runtime assertion check fails.

Regression Testing In addition to unit and integration tests written to reflect the specifications of a system, new tests should be added whenever a flaw is uncovered and fixed. Flaws tend to recur in software for a variety of reasons. The existence of the initial flaw implies that there is some subtlety to that particular code, raising the baseline likelihood of flaws. The fix applied to the code may have only fixed the flaw under the limited set of conditions that were observed at the time, for example in a bug report. The fix may also have depended on assumptions about code elsewhere in the project, and the flaw can recur once those assumptions change.

Running a regression test for every flaw in the project's history assures us that those flaws are not present in the current software artifacts, but for a long project, the weight of that history can make the regression suite unfeasibly large. Many longer-term projects therefore split their regression tests into multiple suites: a small, quick suite to run before each code commit, a larger suite to run every night, and sometimes a full suite that runs over weekends or before major project milestones. Since a goal of testing is to uncover flaws as quickly as possible, running tests less frequently is a tradeoff, and prioritizing and minimizing the cost of testing is an area of active research [63].

Integration Testing While unit testing and regression testing detects flaws on a per-module granularity, integration testing detects flaws in the system as a whole. This type of testing can expose flaws in the way multiple modules interact, measure performance of the integrated system, reveal environmental (e.g., configuration, operating system, network) dependencies, and simulate the end-to-end experience of the system's users.

The most basic integration test is simply to check that the complete system can be successfully built. Once built, integration tests exercise substantial functionality across multiple modules, often simulating the actions performed by a user during an interaction with the system and checking for expected outcomes. In this sense, integration tests are frequently the first line of validation applied to a system.

For example, one integration test might load a ballot, make selections, change selections, and then cast the ballot. Another integration test might follow the same steps, but then spoil the ballot and repeat with a new ballot. While each of these individual steps might concern only a single module, the combination of steps helps expose potential problems with modules interacting, for example if a ballot successfully loads but fails to allow the ballot to be marked correctly.

In addition to simulating end-to-end functionality, integration tests also test the suitability of the software in its intended environments. This is critical for systems that are intended to work with multiple operating systems, with or without a network, or with specialized hardware peripherals like a ballot marking device. Making the environmental assumptions explicit in integration tests also helps prevent flaws from arising due to unstated assumptions. For example, a developer

using the latest version of Linux may inadvertently write software that depends on that version of Linux and fails when run on the version of Linux used in the deployed environment. It is counterproductive and often infeasible for developers to use the deployed environment (it may not even be a general-purpose computer), and so the integration tests must accurately recreate that environment.

Continuous Integration

The expense of fixing software flaws increases over time as other parts of a program accumulate over time around those flaws. When a flawed feature is new, developers have not had a chance to write other code that depends on the flawed code. Once those dependencies exist, a fix for a single flaw can have consequences that ripple outward across the entire project, making the fix much more expensive. The cheapest way to fix a flaw, then, is to discover the flaw as quickly as possible.

Continuous integration (CI) and testing facilitate this by discovering flaws as part of a regular, automatic process that is not dependent on due diligence of individual developers. CI interleaves the quality control process into the development process, rather than leaving it as a separate phase for the end of a project after development has finished.

CI tools automatically build and test the latest version of the software in the VCS system on a regular basis, such as every night, or after every VCS commit. Because the software is built from the VCS, it is important for developers to frequently commit their work to the VCS. The VCS integration ensures the tests are always run on the canonical version of the software, and that any discovered flaws can be linked to a particular version in the VCS system. Instead of potentially having to search the entire codebase for the cause of a flaw, isolating the failing version focuses the efforts of developers on the set of changes introduced in that version, saving time.

CI systems substantially replace manual effort and the risk of manual mistakes when releasing software. Since the CI system is building the project on a nightly basis, it can also post the artifacts of those builds for users and testers to quickly adopt. When an official release like "Version 1.0" is ready for release, developers can simply run the CI system to produce the final artifact. Because the same system is responsible for both the continuous testing and validation of the system and the creation of the final release, it is less likely that the final release will have flaws that would have been caught through earlier testing.

Code Review

Code review practices involve examining the results of the software development process to find flaws, identify potential improvements, and increase understanding of the software throughout the engineering team. Reviews are also an opportunity to ensure that organizational code style standards are met, and that the code and its documentation is clear enough that it can be effectively communicated to others during the review. This process, like discovering flaws during testing and investigating issue reports, feeds back into an interative development process to improve the quality of the final product.

Code review can be a manual process at varying levels of rigor. On the formal end, processes like Fagan inspection require a line-by-line inspection by many developers in an extended meeting, and catch a high percentage of flaws [26]. On lightweight end, code review inherently occurs during pair programming, and can take place informally via a developer-led walkthrough or simply an email to colleagues requesting feedback. Formal inspections are more costly than informal reviews, but may be more suitable for projects that require concrete audit trails for accountability. Lightweight methods, particularly pair programming, can find similar proportions of flaws for lower cost [56] and has other knock-on effects such as higher developer job satisfaction and improved team dynamics [22].

Automated tools complement any form of manual code review. Lightweight static analysis tools and code "lint" tools can help developers avoid common coding mistakes and adhere to organizational style standards. Very lightweight static tools can be run by individual developers before committing their work to the VCS, and longer-running analyses can be part of the continuous integration and testing process. Such analyses are not substitutes for formal verification, however, as they typically are meant to discover small-scale defects and help developers avoid common pitfalls rather than proving overall properties about the correctness of a system.

Release Management & Lifecycle

Release management is the process through which software moves from implementation through testing, validation, and verification into a finished product that can be used in its intended environment. Release management is primarily focused on the smooth integration of the different aspects of the project, and on adhering to practices that make releases repeatable, reliable, and auditable.

Release management and VCS workflows are tightly connected. For example, release managent would be responsible for creating a new release branch in the Git Flow model [7], imposing a feature freeze (no new features, only bug fixes) on that branch, and eventually tagging that branch upon release and merging it back into the main development branch.

For a project that delivers software as a service on a web server, release management would be responsible for deploying the software to production servers. For software delivered as a binary download or CD, release management would be responsible for cryptographically signing and distributing the binary. In both of these cases, the release manager serves as the final line of quality assurance before the software is used in its intended environment, and so she must be fluent enough with all aspects of the project and its processes to release software only once the processes have been faithfully executed.

Testable Documentation

Documentation of the design, implementation, and use of a software system is a standard requirement in software engineering methodologies. However when a system is under development and rapidly changing, documentation can lag behind and fall out of step with the latest version of the software, leading to errors and confusion.

Where possible, documentation should be machine-testable (or even machine-generated) and integrated with the VCS rather than being a set of static resources maintained independently of the software itself. Much as testing gives the advantage of early software flaw detection, testable and generated documentation is less likely to become inconsistent with the software it describes.

The form of testable documentation varies depending on the granularity of the documentation and the underlying technologies used by the project. For example, Business Object Notation (BON) [59] can be used as analyzable documentation at the specification, design, and architecture level.

At the level of code modules and interfaces, documentation should be concretely executable like the "doctest" features available for programming languages like Python and Haskell [27]. Documentation in this style contains short examples that illustrate the expected use of a system and its expected response, for example in Python:

```
"""
This is the fibonacci module. It provides the function fib which
returns the nth fibonacci number, where n >= 0.

>>> fib(0)
0
>>> fib(10)
55
>>> fib(-1)
Traceback (most recent call last):
    ...
ValueError: n must be >= 0
"""
```

These executable tests should supplement, not replace traditional prose documentation. Since they amount to a form of unit test, they suffer from the same limitations. They usually only exercise a handful of concrete values, and cannot test non-functional properties like expected performance or thread safety.

Reproducibility & Automation

A team can implement many of the processes in this section manually. Tests can be run by hand on developers' machines, code can be sent out for review by email, issues can be tracked in mailing list posts, and a release manager can build and package release artifacts by hand for each supported platform. Each time a step in a process must be manually performed, the probability of human error increases, and reproducing steps for later quality assurance and troubleshooting becomes harder.

A manual operator might skip a step or perform a step out of order, for example running the test suite before integrating the latest changes from the VCS. The operator might also introduce new steps that seem necessary and obvious, but unless recorded will make it very difficult to reproduce or audit the process in the future. Finally, manual execution of a process, even if done correctly, takes much longer than automated execution.

To prevent errors, improve reproducibility, and make development more efficient, processes should be automated as much as possible. The methodologies described in this section all support automation and reproducibility or can themselves be automated to a degree, but some play key roles:

Version Control The version control system is a linchpin of automation on a project. The versions it manages are the starting point for automated and reproducible continuous integration, testing, and software releases. The VCS can itself trigger automated processes, for example it could trigger a run of the test suite after every commit.

Any automated process should be run in the context of a particular VCS version, and any artifacts produced by these processes should refer to this version. For example, if software has a built-in bug reporting feature, those reports should automatically include the version at the time the software was built. This allows engineers to easily reproduce the exact circumstances where the user discovered a flaw.

Version control can only improve reproducibility when all of the relevant inputs to a process are managed in the VCS. For example, a software build process that depends on a configuration file in the user's home directory would not be reproducible on a different computer without that home directory.

Testing Manual testing can play an important role when evaluating a system, but any realistic system requires more tests than are feasible to perform manually. Even if the contents of a test suite are automatic, if that test suite is only run manually it will often be skipped, particularly when it takes a long time to run. Automating both the tests themselves and the running of those tests ensures that they will be run on a consistent basis, and that the results will be reproducible and traceable to a particular version.

Continuous Integration Continuous integration is another linchpin of project automation. Since CI tools are designed to automatically run on a regular basis and offer integration with the VCS, other processes are usually automated by using these tools. For example, after building the integrated software, a CI tool should run the test suite and archive the built artifacts for subsequent release management.

Release Management No process has as many moving parts or cross-project concerns as release management, making manual release management extremely error-prone. The entire process from checking out a version from the VCS to deploying the final release artifacts should be as automatic as possible. The manual intervention should amount to simply deciding which version to release, and checking before the final release that the automated process performed as expected.

Because automation is inexpensive when using continuous integration, it is a good practice to have CI tools perform parts of the release management process on a regular basis, even when software is not ready for a release. If the process of producing a release is the same as performing an ordinary nightly build and test, it is less likely that problems will arise only at the release stage when it is much more costly to address those problems.

8.4 Technologies

8.5 Evidence-based Elections Technology

8.5.1 Measuring and Assessing Quality

8.5.2 Interpreting Evidence for the Non-expert

Chapter 9

Feasibility (*Poorvi et al.*) (25%)

9.1 Threats and Security Risks

9.2 Availability

19: Should this be "accessibility"?

9.3 Usability

9.4 Legal Frameworks and Politics

9.5 LEO Considerations

9.6 Cost

9.6.1 Design and Development

9.6.2 Operational

9.6.3 Integration with Local Election Systems and Processes

Chapter 10

Conclusion (Joe K./Susan) (0%)

10.1   Results

10.2   Recommendation (YES or NO???)

10.3   Next Steps

10.3.1   Political/Legal Challenges

10.3.2   Research Challenges

10.3.3   Engineering Challenges

10.3.4   Business Opportunities

Appendix A

BON Representation of E2E VIV Requirements (Dan/Joe K.) (50%)

BON (from external files) will appear here. Currently it is just dumped in a somewhat reasonable order, but it will be cleaned up and brought up to date.

```
scenario_chart E2EVIV_REQUIREMENTS
indexing
  title: "Requirements for End-to-end Verifiable Internet Voting Systems.";
  editor: "Joe Kiniry <kiniry@galois.com>", "Daniel M. Zimmerman <dmz@galois.com>";
  created: "16 July 2014";
  revised: "April 2015"
explanation
  "Functional and non-functional requirements for end-to-end \
 \ verifiable internet voting systems.  Requirements consisting of two \
 \ or more sentences are in fact stipulating multiple, related \
 \ requiremenets in a since scenario.  We index requirements from one, \
 \ thus SYSTEM_AND_DATA_ACCESS_CONTROL requirement 1 is 'Only persons \
 \ appointed by the electoral authority shall have access to the \
 \ central infrastructure, the servers and the election data.'."
end

scenario_chart TECHNICAL_REQUIREMENTS
indexing
  partof: "E2EVIV_REQUIREMENTS";
explanation
  "General technical requirements for digital elections systems."
end

scenario_chart NON_FUNCTIONAL_REQUIREMENTS
indexing
  partof: "E2EVIV_REQUIREMENTS"
explanation
  "General non-functional requirements of digital voting systems."
end

scenario_chart ACCESSIBILITY_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General requirements for accessibility of digital election systems."
```

```
scenario
  "MANDATORY_ACCESSIBILITY_TESTING" -- @ref Kiniry/Zimmerman
description
  "Accessibility testing for disabled and abled voters shall be performed, \
 \ and the reports of the testing made public. The system must achieve \
 \ satisfactory accessibility testing results before being used in a \
 \ binding election."

-- @ref Rec(2004)11 Accessibility
scenario "UNIVERSAL_ACCESSIBILITY" -- @ref Rec(2004)11 Appendix III, A. 61.
description
  "Measures shall be taken to ensure that the relevant software and \
 \ services can be used by all voters and, if necessary, provide access \
 \ to alternative ways of voting."

scenario "ACCESSIBILITY_STAKEHOLDERS" -- @ref Rec(2004)11 Appendix III, A. 62.
description
  "Users shall be involved in the design of e-voting systems, \
 \ particularly to identify constraints and test ease of use at each \
 \ main stage of the development process."

scenario "USER_FACILITIES_FOR_ACCESSIBILITY" -- @ref Rec(2004)11 Appendix III, A. 63.
description
  "Users shall be supplied, whenever required and possible, with \
 \ additional facilities, such as special interfaces or other \
 \ equivalent resources, such as personal assistance."

scenario "COMPLEMENT_ACCESSIBILITY_TECHNOLOGIES" -- @ref Rec(2004)11 Appendix III, A.
    64.
description
  "Consideration shall be given, when developing new products, to \
 \ their compatibility with existing ones, including those using \
 \ technologies designed to help people with disabilities."

scenario "ACCESSIBLE_VOTING_OPTIONS" -- @ref Rec(2004)11 Appendix III, A. 65.
description
  "The presentation of the voting options shall be optimised for the \
 \ voter."
end

scenario_chart ASSURANCE_REQUIREMENTS
indexing
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "General non-functional assurance requirements which increase system \
 \ and election assurance."

scenario
  "CLIENT_ENVIRONMENTS" -- @ref David Jefferson
description
  "Client side software (applications, apps, scripts, etc.) should be \
 \ free of known bugs on a wide range of platform and software stack \
 \ combinations intended to be usable as voting terminals."

scenario
  "AUTHENTICATION_RESILIENCE" -- @ref David Jefferson
description
  "There must be no way to automate forging or invalidation of \
 \ voter authentications without compromising the cryptographic \
```

```
  \ protocols or secrets used in the system."

scenario
  "OPEN_DOCUMENTATION" -- @ref David Jefferson
description
  "All aspects of the design, architecture, algorithms and \
 \ documentation for the entire Internet voting system (not just the \
 \ E2EV core) should be published and available for free download by \
 \ anyone."

scenario
  "DOCUMENTATION_CONSISTENCY" -- @ref David Jefferson
description
  "As the system changes, all documentation must be kept up to \
 \ date.  No new version of an E2EV Internet voting system may be \
 \ certified until all documentation is up to date."

scenario
  "OPEN_SOURCE" -- @ref David Jefferson
description
  "The source code, build scripts, issue tracking system, security \
 \ features, and related development information for the entire \
 \ Internet voting system (all versions for all platforms) shall be \
 \ made publicly available for free download and inspection by \
 \ anyone."

scenario
  "SOURCE_LICENSE" -- @ref David Jefferson
description
  "The source code for all parts of the E2EV Internet voting system \
 \ shall be made publicly available under a license that permits \
 \ anyone to download the code and build, instrument, and test it."

end

scenario_chart AUDITING_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General requirements pertaining to auditing systems and digital \
 \ election systems."

-- @ref Rec(2004)11 Audit, I. General
scenario "AUDIT_SYSTEMS" -- @ref Rec(2004)11 Appendix III, E. I. 100.
description
  "The audit system shall be designed and implemented as part of the \
 \ e-voting system.  Audit facilities shall be present on different \
 \ levels of the system: logical, technical and application."

scenario "AUDITING_COMPLETENESS" -- @ref Rec(2004)11 Appendix III, E. I. 101.
description
  "End-to-end auditing of an e-voting system shall include recording, \
 \ providing monitoring facilities and providing verification \
 \ facilities."

-- @ref Rec(2004)11 Audit, II. Recording
scenario "AUDIT_SYSTEM_BASELINE" -- @ref Rec(2004)11 Appendix III, E. II. 102.
description
  "The audit system shall be open and comprehensive, and actively \
```

```
 \ report on potential issues and threats."

scenario "AUDIT_SYSTEM_DATA" -- @ref Rec(2004)11 Appendix III, E. II. 103.
description
  "The audit system shall record times, events and actions, including: \
 \ a. all voting-related information, including the number of eligible \
 \ voters, the number of votes cast, the number of invalid votes, the \
 \ counts and recounts, etc.; b. any attacks on the operation of the \
 \ e-voting system and its communications infrastructure; c. system \
 \ failures, malfunctions and other threats to the system."

-- @ref Rec(2004)11 Audit, III. Monitoring
scenario "AUDIT_SYSTEM_EVIDENCE" -- @ref Rec(2004)11 Appendix III, E. III. 104.
description
  "The audit system shall provide the ability to oversee the election \
 \ or referendum and to verify that the results and procedures are in \
 \ accordance with the applicable legal provisions."

scenario "AUDIT_DATA_SECURITY" -- @ref Rec(2004)11 Appendix III, E. IIi. 105.
description
  "Disclosure of the audit information to unauthorized persons shall \
 \ be prevented."

scenario "AUDIT_DATA_SECRECY" -- @ref Rec(2004)11 Appendix III, E. III. 106.
description
  "The audit system shall maintain voter anonymity at all times."

-- @ref Rec(2004)11 Audit, II. Verifiability
scenario "AUDIT_SYSTEM_CAPABILITY" -- @ref Rec(2004)11 Appendix III, E. IV. 107.
description
  "The audit system shall provide the ability to cross-check and \
 \ verify the correct operation of the e-voting system and the accuracy \
 \ of the result, to detect voter fraud, and to prove that all counted \
 \ votes are authentic and that all votes have been counted."

scenario "AUDIT_SYSTEM_FOR_LEGAL_COMPLIANCE" -- @ref Rec(2004)11 Appendix III, E. IV.
    108.
description
  "The audit system shall provide the ability to verify that an \
 \ e-election or e-referendum has complied with the applicable legal \
 \ provisions."

-- @ref Rec(2004)11 Audit, II. Other
scenario "AUDIT_DATA_VALIDITY" -- @ref Rec(2004)11 Appendix III, E. V. 109.
description
  "The audit system shall be protected against attacks that may \
 \ corrupt, alter or lose records in the audit system."

scenario "AUDIT_DATA_CONFIDENTIALITY" -- @ref Rec(2004)11 Appendix III, E. V. 110.
description
  "The electoral authority shall take adequate steps to ensure that the \
 \ confidentiality of any information obtained by any person while \
 \ carrying out auditing functions is guaranteed."

scenario
  "LOG_BASICS" -- @ref David Jefferson
description
  "The Internet voting system should keep detailed logs of all \
 \ relevant activity."
```

```
scenario
  "LOG_IMMUTABILITY" -- @ref David Jefferson
description
  "Log entries must be unmodifiable once written."

scenario
  "LOG_COMMITMENT" -- @ref Ron Rivest
description
  "Log entries must accurately reflect the commitment character \
 \ of elections and the relationships among election events \
 \ (e.g., ballot, vote, voter, and election state transitions)."

scenario
  "LOG_DATA_COMPLETENESS" -- @ref David Jefferson
description
  "The log data should be as complete as possible, consistent with \
 \ maximum possible vote privacy."

scenario
  "PRIVACY_VS_FRAUD_TRADEOFF" -- @ref David Jefferson
description
  "If there is a tradeoff between vote privacy and the identification \
 \ of the perpetrators of fraud, the decision should be made in favor \
 \ of vote privacy."

scenario
  "VOTER_LIST" -- @ref David Jefferson
description
  "The list of voters who voted online should be published."
end

scenario_chart AUDITING_REQUIREMENTS_VERIFICATION
indexing
  partof: "AUDITING_REQUIREMENTS"
explanation
  "Requirements specific to auditing verifiable elections."

scenario
  "VERIFICATION_PARTIAL_FAILURE" -- @ref David Jefferson
description
  "The system, in the event that it does not verify the online \
 \ votes cast, must be capable of giving an upper bound on the \
 \ number of ballots that may have been affected."

scenario
  "VERIFICATION_SOURCE" -- @ref David Jefferson
description
  "Official verification applications, like the voting software itself, \
 \ must be published in source form along with documentation, build \
 \ directions, and a standard cryptographic hash of the source code."
end

scenario_chart AUTHENTICATION_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General requirements relating to the authentication of principles \
 \ (both computers and humans) involved in any digital election \
```

```
  \ system."

scenario
  "VOTER_AUTHENTICATION" -- @ref David Jefferson
description
  "The voting service must by itself securely authenticate the voter \
 \ (verify identify the voter and verify his/her registration and/or \
 \ eligibility according to law to vote in the election) before \
 \ allowing him/her to cast a ballot (or modify or replace a \
 \ previously cast ballot)."

scenario
  "NO_THIRD_PARTY_AUTHENTICATION" -- @ref David Jefferson
description
  "Authentication must not be done through third party intermediaries \
 \ such as Facebook, iCloud, Google, Yahoo, Amazon, etc. that offer \
 \ authentication services."

scenario
  "SECRET_AUTHENTICATION_SHARED_SECRETS" -- @ref David Jefferson
description
  "Authentication for remote voting systems must not use personal \
 \ information, government or commercial account identifiers, etc."

scenario
  "AUTHENTICATION_DATA_UPDATES" -- @ref David Jefferson
description
  "Authentication secrets must be changeable or revokable at \
 \ any time at the behest of either the voter or election \
 \ officials."

scenario
  "AUTHENTICATION_DATA_REFRESH_PERIODICITY" -- @ref David Jefferson
description
  "All voter authentication secrets must be changed at least once in \
 \ every election cycle."

end

scenario_chart CERTIFICATION_FUNCTIONAL_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "Requirements relating to the functional certification of digital \
 \ election systems and elections."

scenario "AUTOMATED_TESTING" -- @ref Kiniry/Zimmerman
description
  "Each functional requirement must have an associated set of automated \
 \ tests that provide evidence that the requirement is fulfilled."

scenario "ELECTION_PROTOCOL_PROOFS" -- @ref Kiniry/Zimmerman
description
  "The election protocol shall have associated formal proofs of correctness \
 \ and security."
end

scenario_chart CERTIFICATION_NON_FUNCTIONAL_REQUIREMENTS
indexing
```

```
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "Requirements relating to the non-functional certification of \
 \ election systems and elections."

-- @ref Rec(2004)11 Certification
scenario "CERTIFICATION_PROCESSES" -- @ref Rec(2004)11 Appendix III, F. 111.
description
  "The electoral authority shall introduce certification processes that allow \
 \ for any ICT (Information and Communication Technology) component to \
 \ be tested and certified as being in conformity with technical \
 \ requirements."

scenario
  "CERTIFICATION_PARTIES_COMPETENCE" -- @ref David Jefferson
description
  "Any E2EV Internet voting system should be certified by competent \
 \ professionals."

scenario
  "CERTIFICATION_REPORT_TRANSPARENCY" -- @ref David Jefferson
description
  "Any and all certification reports issued by certification \
 \ professionals must be public, whether they recommend \
 \ certification or not."

scenario
  "RECERTIFICATION_CONDITIONS" -- @ref David Jefferson
description
  "Any time there is a change in the voting system client or server \
 \ side or the E2EV system, all of the requirements must \
 \ be re-established and recertified.  Changes that mandate \
 \ re-certification include, but are not limited to: new supported \
 \ hardware platforms, OS's, browsers, etc.; bug fixes and security \
 \ patches to voting client and/or server; changes or upgrades to \
 \ voting client or server in response to detected bugs or security \
 \ vulnerabilities, changes in law, or changes in threat environment."

scenario
  "RECERTIFICATION_PERIODICITY" -- @ref David Jefferson
description
  "The requirements must be re-established and recertified every \
 \ election cycle even if there are no changes."

scenario
  "VALIDATION_PLATFORM_COVERAGE" -- @ref David Jefferson
description
  "The system must be extensively tested on a wide range of platform \
 \ and software combinations."

scenario
  "PUBLIC_VALIDATION_PLATFORM_COVERAGE_RESULTS" -- @ref David Jefferson
description
  "All test procedures and results for platform coverage must be public."

end

scenario_chart EVOLVABILITY_REQUIREMENTS
indexing
```

```
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "General requirements on the evolvability of digital election \
 \ systems."

scenario
  "ELECTORAL_AUTHORITY_UPDATE"
description
  "The electoral authority has the right and ability to update \
 \ election systems to conform to changes in applicable law, \
 \ available techology, or the system threat model."

end

scenario_chart FUNCTIONAL_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General functional requirements for digital election systems."

scenario
  "CASTING_ATOMIC" -- @ref David Jefferson
description
  "Ballot casting shall be atomic with respect to server failures."

scenario
  "DETERMINISTIC_VOTING_PROCESS" -- @ref David Jefferson
description
  "If a server side failure occurs, no voter's balloting can be \
 \ left in an unknown state."

scenario
  "BALLOT_FINAL_STATES" -- @ref David Jefferson
description
  "Either a ballot is securely and completely cast and the \
 \ voter is marked as having voted, or no ballot is recorded and the \
 \ voter is not marked as having voted."

scenario
  "VOTE_RECORD_MONOTONICITY" -- @ref David Jefferson
description
  "If the system and the law allows a voter to cast multiple votes \
 \ with only the last one counting, or to cast a partial ballot with \
 \ the option of modifying it later, then each voting session must be \
 \ atomic with respect to server failures. If a failure occurs during the \
 \ voter's last session, then the votes cast as of his or her previous \
 \ session will count."

scenario
  "RECEIPT_FREEDOM" -- @ref David Jefferson
description
  "There must be no way for voters to prove to another party any \
 \ information regarding how they voted in any race (beyond what is \
 \ mathematically deducible from the final distribution of votes)."

scenario
  "VALID_BALLOT_PROVENANCE" -- @ref David Jefferson
description
  "Once it is determined that a ballot will be counted, the ballot \
```

```
  \ shall be irrevocably separated from the identification of the \
  \ voter who cast it."

scenario
  "MULTI_BALLOT_RECORD" -- @ref David Jefferson
description
  "If the voting system permits voters to modify or replace their \
  \ previously cast ballots, only the latest vote by each voter in \
  \ each race shall be counted in the final tally."

scenario
  "NO_DOUBLE_VOTE" -- @ref David Jefferson
description
  "But for systems supporting MULTI_BALLOT_RECORD, the voting system \
  \ shall not record more than one vote for any voter in any race."

scenario
  "NO_ADVERTISING" -- @ref David Jefferson
description
  "The voting system client must not display or permit the display of \
  \ any advertising or commercial logos in the window that contains the \
  \ voting session, other than those of the election jurisdiction \
  \ itself."

scenario
  "NO_EXTERNAL_LINKS" -- @ref David Jefferson
description
  "The voting system client must not display any links to other sites \
  \ except for help in the mechanics of voting."

end

scenario_chart INTEROPERABILTY_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General requirements on the interoperability of digital election \
  \ systems."

-- @ref Rec(2004)11 Interoperabilty
scenario "OPEN_STANDARDS" -- @ref Rec(2004)11 Appendix III, B. 66.
description
  "Open standards shall be used to ensure that the various technical \
  \ components or services of an e-voting system, possibly derived \
  \ from a variety of sources, interoperate."

scenario "EML" -- @ref Rec(2004)11 Appendix III, B. 67.
description
  "The Election Markup Language (EML) shall be used whenever possible \
  \ for e-election and e-referendum applications."

scenario "DATA_LOCALIZATION" -- @ref Rec(2004)11 Appendix III, B. 68.
description
  "In cases that imply specific election or referendum data \
  \ requirements, a localization procedure shall be used to accommodate \
  \ these needs."

scenario
  "OPEN_LOG_FORMATS" -- @ref David Jefferson
```

```
description
  "The log data and documentation of its meaning and format shall be \
 \ available for public download so that anyone can download, inspect, \
 \ and publish concerns based on the logs."
end

scenario_chart LEGAL_REQUIREMENTS
indexing
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "General legal requirements relating to legal matters and digital \
 \ election systems."

-- @ref Rec(2004)11 Universal Suffrage
scenario -- @ref Rec(2004)11 Appendix I, A. I. 1.
  "USABLE_UI"
description
  "The voter interface of an e-voting system shall be understandable and \
 \ easily usable."

scenario -- @ref Rec(2004)11 Appendix I, A. I. 2.
  "UNIMPEDED_REGISTRATION"
description
  "Possible registration requirements for e-voting shall not pose \
 \ an impediment to the voter participating in e-voting."

scenario -- @ref Rec(2004)11 Appendix I, A. I. 3.
  "MAXIMIZE_DISABLED_ACCESSIBILITY"
description
  "E-voting systems shall be designed, as far as it is practicable, to \
 \ maximize the opportunities that such systems can provide for persons \
 \ with disabilities."

scenario -- @ref Rec(2004)11 Appendix I, A. I. 4.
  "REMOTE_ONLY_SUPPLEMENTARY"
description
  "Unless channels of remote e-voting are universally accessible, they \
 \ shall be only an additional and optional means of voting."

-- @ref Rec(2004)11 Equal suffrage
scenario -- @ref Rec(2004)11 Appendix I, A. II. 5--6.
  "AT_MOST_ONE_VOTE_PER_VOTER"
description
  "The e-voting system shall ensure that at most one electronic vote from \
 \ each voter is included in the final tally."

scenario -- @ref Rec(2004)11 Appendix I, A. II. 7.
  "VALID_TALLY"
description
  "Every vote deposited in an electronic ballot box shall be counted, and \
 \ each vote cast in the election or referendum shall be counted only once."

scenario -- @ref Rec(2004)11 Appendix I, A. II. 8.
  "VOTE_AGGREGATION"
description
  "Where electronic and non-electronic voting channels are used in the same \
 \ election or referendum, there shall be a secure and reliable method to \
 \ aggregate all votes and to calculate the correct result."
```

```
-- @ref Rec(2004)11 Free suffrage
scenario -- @ref Rec(2004)11 Appendix I, A. III. 9.
  "FREE_SUFFRAGE"
description
  "The organization of e-voting shall secure the free formation and \
 \ expression of the voter's opinion and, where required, the \
 \ personal exercise of the right to vote."

scenario -- @ref Rec(2004)11 Appendix I, A. III. 10.
  "REFLECTIVE_VOTING_PROCESS"
description
  "The way in which voters are guided through the e-voting process \
 \ shall be such as to prevent their voting precipitately or without \
 \ reflection."

scenario -- @ref Rec(2004)11 Appendix I, A. III. 11.
  "FLEXIBLE_VOTING_PROCESS"
description
  "Voters shall be able to alter their choice at any point in the \
 \ e-voting process before casting their vote, or to break off the \
 \ procedure, without their previous choices being recorded or made \
 \ available to any other person."

scenario -- @ref Rec(2004)11 Appendix I, A. III. 12.
  "NO_VOTER_MANIPULATION"
description
  "The e-voting system shall not permit any manipulative influence to \
 \ be exercised over the voter during the voting."

scenario -- @ref Rec(2004)11 Appendix I, A. III. 13.
  "BLANK_VOTE"
description
  "The e-voting system shall provide the voter with a means of \
 \ participating in an election or referendum without the voter \
 \ exercising a preference for any of the voting options, for example, \
 \ by casting a blank vote."

scenario -- @ref Rec(2004)11 Appendix I, A. III. 14.
  "CONCLUSION_OF_VOTING_PROCESS"
description
  "The e-voting system shall indicate clearly to the voter when the \
 \ vote has been cast successfully and when the whole voting procedure \
 \ has been completed."

scenario -- @ref Rec(2004)11 Appendix I, A. III. 15.
  "IMMUTABLE_VOTES"
description
  "Except in systems supporting MULTI_BALLOT_RECORD, the e-voting system \
 \ shall prevent the changing of a vote once that vote has been cast."

-- @ref Rec(2004)11 Secret suffrage
scenario -- @ref Rec(2004)11 Appendix I, A. IV. 16.
  "SECRET_SUFFRAGE"
description
  "E-voting shall be organized in such a way as to exclude at any \
 \ stage of the voting procedure and, in particular, at voter \
 \ authentication, anything that would endanger the secrecy of the \
 \ vote."
```

```
scenario -- @ref Rec(2004)11 Appendix I, A. IV. 17.
  "ANONYMOUS_VOTES"
description
  "The e-voting system shall guarantee that votes in the electronic \
 \ ballot box and votes being counted are, and will remain, anonymous, \
 \ and that it is not possible to reconstruct a link between the vote \
 \ and the voter."

scenario -- @ref Rec(2004)11 Appendix I, A. IV. 18.
  "NO_INDIRECT_SECRECY_VIOLATION"
description
  "The e-voting system shall be so designed that the expected number \
 \ of votes in any electronic ballot box will not allow the result to \
 \ be linked to individual voters."

scenario -- @ref Rec(2004)11 Appendix I, A. IV. 19.
  "NO_SECRET_SUFFRAGE_SIDE_CHANNEL"
description
  "Measures shall be taken to ensure that the information needed \
 \ during electronic processing cannot be used to breach the secrecy of \
 \ the vote."

scenario
  "NO_NDAS_FOR_STUDY" -- @ref David Jefferson 22-6-2014
description
  "No nondisclosure agreement or any other contract shall be required \
 \ to download and study the Internet voting system."

scenario
  "NO_NDAS_FOR_AUDIT" -- @ref David Jefferson 22-6-2014
description
  "No nondisclosure agreement or any other contract shall be required \
 \ to download, instrument, build, test, and publish test results for \
 \ an E2EV Internet voting system."

end

scenario_chart MAINTENANCE_REQUIREMENTS
indexing
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "General requirements relating to the maintainence of digital election \
 \ systems."

scenario
  "ELECTORAL_AUTHORITY_PATCH"
description
  "The electoral authority has the right and ability to patch \
 \ election systems to correct flaws discovered in the algorithms, \
 \ implementation, or deployment."

end

scenario_chart OPERATIONAL_REQUIREMENTS
indexing
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "General operational requirements for digital election systems."
```

```
-- @ref Rec(2004)11 Notification
scenario "ELECTION_TIMETABLES" -- @ref Rec(2004)11 Appendix II, I. 36.
description
  "Domestic legal provisions governing an e-election or e-referendum \
 \ shall provide for clear timetables concerning all stages of the \
 \ election or referendum, both before and after the election or \
 \ referendum."

scenario "ELECTION_PERIOD" -- @ref Rec(2004)11 Appendix II, I. 37.
description
  "The period in which an electronic vote can be cast shall not begin \
 \ before the notification of an election or a referendum.  Particularly \
 \ with regard to remote e-voting, the period shall be defined and made \
 \ known to the public well in advance of the start of voting."

scenario "EVOTING_OUTREACH" -- @ref Rec(2004)11 Appendix II, I. 38.
description
  "The voters shall be informed, well in advance of the start of \
 \ voting, in clear and simple language, of the way in which the \
 \ e-voting will be organised, and any steps a voter may have to take \
 \ in order to participate and vote."

-- @ref Rec(2004)11 Voters
scenario "VOTER_VERIFIABLE_VOTER_REGISTER" -- @ref Rec(2004)11 Appendix II, II. 39.
description
  "There shall be a voters' register that is regularly updated.  The \
 \ voter shall be able to check, as a minimum, the information that is \
 \ held about him/her on the register, and request corrections."

scenario "ONLINE_VOTER_REGISTER" -- @ref Rec(2004)11 Appendix II, II. 40.
description
  "The possibility of creating an electronic register and introducing \
 \ a mechanism allowing online application for voter registration \
 \ and, if applicable, for application to use e-voting, shall be \
 \ considered.  If participation in e-voting requires a separate \
 \ application by the voter and/or additional steps, an electronic, \
 \ and, where possible, interactive procedure shall be considered."

scenario "VOTER_REGISTRATION_ELECTION_OVERLAP" -- @ref Rec(2004)11 Appendix II, II.
    41.
description
  "In cases where there is an overlap between the period for voter \
 \ registration and the voting period, provision for appropriate \
 \ voter authentication shall be made."

-- @ref Rec(2004)11 Candidates
scenario "ONLINE_CANDIDATE_NOMINATION" -- @ref Rec(2004)11 Appendix II, III. 42.
description
  "The possibility of introducing online candidate nomination may be \
 \ considered."

scenario "PUBLIC_CANDIDATE_LIST" -- @ref Rec(2004)11 Appendix II, III. 43.
description
  "A list of candidates that is generated and made available \
 \ electronically shall also be publicly available by other means."

-- @ref Rec(2004)11 Voting
scenario "MULTIPLE_CHANNELS_ONE_VOTE" -- @ref Rec(2004)11 Appendix II, IV. 44.
description
```

```
   "Where remote e-voting takes place while polling stations are open, \
 \ the system shall be so designed that it prevents any voter from \
 \ voting more than once."

scenario "VOTING_PERIOD_INVARIANT" -- @ref Rec(2004)11 Appendix II, IV. 45.
description
   "Remote e-voting may start and/or end at an earlier time than the \
 \ opening of any polling station.  Remote e-voting shall not continue \
 \ after the end of the voting period at polling stations."

scenario "UNIVERSAL_VOTER_HELP" -- @ref Rec(2004)11 Appendix II, IV. 46.
description
   "For every e-voting channel, support and guidance arrangements on \
 \ voting procedures shall be set up for, and be available to, the \
 \ voter.  In the case of remote e-voting, such arrangements shall also \
 \ be available through a different, widely-available communication \
 \ channel."

scenario "FAIR_VOTING_OPTIONS" -- @ref Rec(2004)11 Appendix II, IV. 47.
description
   "There shall be equality in the manner of presentation of all voting \
 \ options on the device used for casting an electronic vote."

scenario "VOTING_OPTIONS_ONLY" -- @ref Rec(2004)11 Appendix II, IV. 48.
description
   "The electronic ballot by which an electronic vote is cast shall be \
 \ free from any information about voting options, other than that \
 \ strictly required for casting the vote.  The e-voting system shall \
 \ avoid the display of other messages that may influence the voters' \
 \ choice."

scenario "FAIR_VOTING_OPTION_INFORMATION" -- @ref Rec(2004)11 Appendix II, IV. 49.
description
   "If it is decided that information about voting options will be \
 \ accessible from the e-voting site, this information shall be \
 \ presented with equality."

scenario "BINDING_ELECTION_CLARITY" -- @ref Rec(2004)11 Appendix II, IV. 50.
description
   "Before casting a vote using a remote e-voting system, voters' \
 \ attention shall be explicitly drawn to the fact that the e-election \
 \ or e-referendum in which they are submitting their decision by \
 \ electronic means is a real election or referendum.  In case of \
 \ tests, participants shall have their attention drawn explicitly to \
 \ the fact that they are not participating in a real election or \
 \ referendum and shall, when tests are continued at election times, \
 \ at the same time be invited to cast their ballot by the voting \
 \ channel(s) available for that purpose."

scenario "REMOTE_RECEIPT_FREEDOM" -- @ref Rec(2004)11 Appendix II, IV. 51.
description
   "A remote e-voting system shall not enable the voter to be in \
 \ possession of a proof of the content of the vote cast."

scenario "SUPERVISED_VOTE_RECEIPT_FREEDOM" -- @ref Rec(2004)11 Appendix II, IV. 52.
description
   "In a supervised environment, the information on the vote shall \
 \ disappear from the visual, audio or tactile display used by the \
 \ voter to cast the vote as soon as it has been cast.  Where a paper \
```

```
 \ proof of the electronic vote is provided to the voter at a polling \
 \ station, the voter shall not be able to show it to any other per- \
 \ son, or take this proof outside of the polling station."

-- @ref Rec(2004)11 Results
scenario "SECRET_INTERMEDIATE_TALLY" -- @ref Rec(2004)11 Appendix II, V. 53.
description
  "The e-voting system shall not allow the disclosure of the number of \
 \ votes cast for any voting option until after the closure of the \
 \ electronic ballot box.  This information shall not be disclosed to \
 \ the public until after the end of the voting period."

scenario "NO_ITALIAN_ATTACK" -- @ref Rec(2004)11 Appendix II, V. 54.
description
  "The e-voting system shall prevent processing information on votes \
 \ cast within deliberately chosen sub-units that could reveal \
 \ individual voters' choices."

scenario "DECODING_LATENCY" -- @ref Rec(2004)11 Appendix II, V. 55.
description
  "Any decoding required for the counting of the votes shall be \
 \ carried out as soon as practicable after the closure of the voting \
 \ period."

scenario "TALLY_OBSERVATION" -- @ref Rec(2004)11 Appendix II, V. 56.
description
  "When counting the votes, representatives of the competent electoral \
 \ authority shall be able to participate in, and any observers able to \
 \ observe, the count."

scenario "TALLY_RECORD" -- @ref Rec(2004)11 Appendix II, V. 57.
description
  "A record of the counting process of the electronic votes shall be \
 \ kept, including information about the start and end of, and the \
 \ persons involved in, the count."

scenario "INTEGRITY_VIOLATION_RECORD" -- @ref Rec(2004)11 Appendix II, V. 58.
description
  "In the event of any irregularity affecting the integrity of votes, \
 \ the affected votes shall be recorded as having their integrity violated."

-- @ref Rec(2004)11 Audit
scenario "SYSTEM_AUDITABILITY" -- @ref Rec(2004)11 Appendix II, VI. 59.
description
  "The e-voting system shall be auditable."

scenario "SYSTEM_AUDITS_IMPACT" -- @ref Rec(2004)11 Appendix II, VI. 60.
description
  "The conclusions drawn from the audit process shall be applied in \
 \ future elections and referenda."

scenario
  "OPEN_SYSTEM" -- @ref David Jefferson
description
  "The e-voting system must function correctly as an open system, \
 \ where large parts (the mix of client hardware and software in \
 \ fact) are unknown, unsecured, uncertified, and completely out \
 \ of control of election officials."
```

```
scenario
  "SUPPORTED_CLIENTS" -- @ref David Jefferson
description
  "Operators of voting systems must document exactly what client \
 \ configurations are required or supported, including: \
 \  - versions of hardware platforms (PCs, mobile devices, etc.) \
 \  - versions of specific operating systems for those platforms \
 \  - versions of specific browsers, plugins, protocols, or \
 \    other software applications, apps, components, and plugins."

scenario
  "CLIENT_INTERFERENCE" -- @ref David Jefferson
description
  "Operators of voting systems must document exactly which common \
 \ components, plugins, or other software interfere with voting (e.g., \
 \ flash blockers, popup blockers, script blockers, etc.)."

scenario
  "MANDATORY_CLIENT_TECHNOLOGY" -- @ref David Jefferson
description
  "Operators of voting systems must document exactly what configuration \
 \ choices the voter must make to successfully vote (e.g., mandate \
 \ Javascript)."

scenario
  "PRIVACY_ENHANCING_VOTER_OPTIONS" -- @ref David Jefferson
description
  "Operators of voting systems must document exactly what configuration \
 \ choices the voter might wish to make to more strongly protect \
 \ his/her vote privacy; e.g., disable cookies, run privacy-protecting \
 \ browser plugins, vote from virtual machine that is later destroyed, \
 \ log out of social networks, disable remote control and remote \
 \ administration tools, disable incoming connections, etc."

scenario
  "BREADCRUMBS_USER_ADVICE" -- @ref David Jefferson
description
  "Users may be advised to turn off browser history data, cookies, \
 \ logging data, and other tools that might retain a record of the \
 \ vote transaction whether the vote data itself or metadata."

end

scenario_chart PROCEDURAL_REQUIREMENTS
indexing
  partof: "NON_FUNCTIONAL_REQUIREMENTS"
explanation
  "General procedural requirements for digital electoin systems."

-- @ref Rec(2004)11 Transparency
scenario "VOTER_COMPREHENSION_AND_CONFIDENCE" -- @ref Rec(2004)11 Appendix I, B. I.
    20.
description
  "The electoral authority shall take steps to ensure that voters understand and \
 \ have confidence in the e-voting system in use."

scenario "PUBLIC_SYSTEM_FUNCTION" -- @ref Rec(2004)11 Appendix I, B. I. 21.
description
  "Information on the functioning of an e-voting system shall be made \
```

```
  \ publicly available."

scenario "VOTER_PRACTICE" -- @ref Rec(2004)11 Appendix I, B. I. 22.
description
  "Voters shall be provided with an opportunity to practice any new \
 \ method of e-voting before, and separately from, the moment of \
 \ casting an electronic vote."

scenario "OBSERVER_INVARIANTS" -- @ref Rec(2004)11 Appendix I, B. I. 23.
description
  "Any observers, to the extent permitted by law, shall be able to be \
 \ present to observe and comment on the e-elections, including the \
 \ establishing of the results."

-- @ref Rec(2004)11 Verifiability and accountability
scenario "DISCLOSURE_OBLIGATIONS" -- @ref Rec(2004)11 Appendix I, B. II. 24.
description
  "The components of the e-voting system shall be disclosed, at least \
 \ to the competent electoral authorities, as required for verification \
 \ and certification purposes."

scenario "CERTIFICATION_OBLIGATIONS" -- @ref Rec(2004)11 Appendix I, B. II. 25.
description
  "Before any e-voting system is introduced, and at appropriate \
 \ intervals thereafter, and in particular after any changes are made \
 \ to the system, an independent body, appointed by the electoral \
 \ authorities, shall verify that the e-voting system is working \
 \ correctly and that all the necessary security measures have been \
 \ taken."

scenario "RECOUNT_SUPPORTED" -- @ref Rec(2004)11 Appendix I, B. II. 26.
description
  "There shall be the possibility for a recount.  Other features of the \
 \ e-voting system that may influence the correctness of the results \
 \ shall be verifiable."

scenario "RERUN_SUPPORTED" -- @ref Rec(2004)11 Appendix I, B. II. 27.
description
  "The e-voting system shall not prevent the partial or complete \
 \ re-run of an election or a referendum."

-- @ref Rec(2004)11 Reliability and security
scenario "RELIABILITY_AND_SECURITY" -- @ref Rec(2004)11 Appendix I, B. III. 28.
description
  "The electoral authority shall ensure the reliability and \
 \ security of the e-voting system."

scenario "NO_FRAUD_OR_INTERVENTION" -- @ref Rec(2004)11 Appendix I, B. III. 29.
description
  "All possible steps shall be taken to avoid the possibility of fraud \
 \ or unauthorized intervention affecting the system during the whole \
 \ voting process."

scenario "SYSTEM_AVAILABILITY" -- @ref Rec(2004)11 Appendix I, B. III. 30.
description
  "The e-voting system shall contain measures to preserve the \
 \ availability of its services during the e-voting process.  It shall \
 \ resist, in particular, malfunction, breakdowns or denial of service \
 \ attacks."
```

```
scenario "SYSTEM_GENUINE_AND_CORRECT" -- @ref Rec(2004)11 Appendix I, B. III. 31.
description
  "Before any e-election or e-referendum takes place, the competent \
 \ electoral authority shall satisfy itself that the e-voting system \
 \ is genuine and operates correctly."

scenario "SYSTEM_AND_DATA_ACCESS_CONTROL" -- @ref Rec(2004)11 Appendix I, B. III. 32.
description
  "Only persons appointed by the electoral authority shall have access \
 \ to the central infrastructure, the servers and the election \
 \ data.  There shall be clear rules established for such \
 \ appointments.  Critical technical activities shall be carried out by \
 \ teams of at least two people.  The composition of the teams shall be \
 \ regularly changed.  As far as possible, such activities shall be \
 \ carried out outside election periods."

scenario "OPEN_BALLOT_BOX_INVARIANTS" -- @ref Rec(2004)11 Appendix I, B. III. 33.
description
  "While an electronic ballot box is open, any authorised intervention \
 \ affecting the system shall be carried out by teams of at least two \
 \ people, be the subject of a report, and be monitored by \
 \ representatives of the competent electoral authority and any \
 \ election observers."

scenario "VOTES_INVARIANTS" -- @ref Rec(2004)11 Appendix I, B. III. 34.
description
  "The e-voting system shall maintain the availability and integrity \
 \ of the votes.  It shall also maintain the confidentiality of the \
 \ votes and keep them sealed until the counting process.  If stored or \
 \ communicated outside controlled environments, the votes shall be \
 \ encrypted."

scenario "SEALED_VOTES_VOTER_RELATION" -- @ref Rec(2004)11 Appendix I, B. III. 35.
description
  "Votes and voter information shall remain sealed as long as the data \
 \ is held in a manner where they can be associated.  Authentication \
 \ information shall be separated from the voter's decision at a \
 \ pre-defined stage in the e-election or e-referendum."

scenario
  "VERIFICATION_FAILURE_PROCEDURES" -- @ref David Jefferson
description
  "There must be clear technical and legal procedures for how to \
 \ proceed in the event that voters can prove that their votes were \
 \ not received accurately or counted, or if the official election \
 \ verification application does not verify that the Internet part of \
 \ the election was correct."

end

scenario_chart SYSTEM_OPERATIONAL_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General system operational requirements for digital election \
 \ systems."

-- @ref Rec(2004)11 Systems Operation
```

```
scenario "PUBLIC_SYSTEM_MANIFEST" -- @ref derived from Rec(2004)11 Appendix III, C.
    69.
description
  "The electoral authority shall publish an official manifest of the \
 \ software used in an e-election or e-referendum.  It may exclude \
 \ from the public manifest data protection software for security \
 \ reasons. At the very least the manifest shall indicate the software \
 \ used, the versions, its date of installation and a brief description. \
 \ A procedure shall be established for updating the manifest to reflect \
 \ changes to the installed software."

scenario "PRIVATE_SYSTEM_MANIFEST" -- @ref derived from Rec(2004)11 Appendix III, C.
    69.
description
  "The electoral authority shall maintain a manifest of all software, \
 \ including data protection software, used in the system. This manifest \
 \ shall contain at least the same information as the public manifest. \
 \ A procedure shall be established for updating the manifest to reflect \
 \ changes to the installed software."

scenario "MANIFEST_ACCURACY" -- @ref derived from Rec(2004)11 Appendix III, C. 69.
description
  "It shall be possible for the electoral authority to check the installed \
 \ software against the system manifests at any time."

scenario "SYSTEM_FAILOVER_INVARIANTS" -- @ref Rec(2004)11 Appendix III, C. 70.
description
  "Those responsible for operating the equipment shall draw up a \
 \ contingency procedure for system failures.  Any backup system shall \
 \ conform to the same standards and requirements as the original system."

scenario "DATA_BACKUP_INVARIANTS" -- @ref Rec(2004)11 Appendix III, C. 71.
description
  "Sufficient backup arrangements shall be in place and be permanently \
 \ available to ensure that voting proceeds smoothly.  The staff \
 \ concerned shall be ready to intervene rapidly according to a \
 \ procedure drawn up by the electoral authority."

scenario "SYSTEM_INVARIANTS_DURING_ELECTION" -- @ref Rec(2004)11 Appendix III, C. 72.
description
  "Those responsible for the equipment shall use special procedures to \
 \ ensure that during the polling period the voting equipment and its \
 \ use satisfy requirements.  The backup services shall be regularly \
 \ monitored."

scenario "PRE_ELECTION_CERTIFICATION_INVARIANTS" -- @ref Rec(2004)11 Appendix III, C.
    73.
description
  "Before each election or referendum, the equipment shall be checked \
 \ and approved in accordance with a protocol drawn up by the \
 \ electoral authority.  The equipment shall be checked to ensure that \
 \ it complies with technical specifications.  The findings shall be \
 \ submitted to the electoral authority."

scenario "FORMAL_CONTROL_PROCEDURE" -- @ref Rec(2004)11 Appendix III, C. 74.
description
  "All technical operations shall be subject to a formal control \
 \ procedure.  Any substantial changes to key equipment shall be \
 \ performed with advance notice."
```

```
scenario "PHYSICAL_SECURITY_OF_SYSTEMS_INVARIANTS" -- @ref Rec(2004)11 Appendix III, C
    . 75.
description
  "Key e-election or e-referendum equipment shall be located in a \
 \ secure area and that area shall, throughout the election or \
 \ referendum period, be guarded against interference of any sort and \
 \ from any person.  During the election or referendum period a \
 \ physical disaster recovery plan shall be in place.  Furthermore, any \
 \ data retained after the election or referendum period shall be \
 \ stored securely."

scenario "INCIDENT_RESPONSE_INVARIANTS" -- @ref Rec(2004)11 Appendix III, C. 76.
description
  "Where incidents that could threaten the integrity of the system \
 \ occur, those responsible for operating the equipment shall \
 \ immediately inform the electoral authority, which will \
 \ take the necessary steps to mitigate the effects of the \
 \ incident.  The level of incident that shall be reported shall be \
 \ specified in advance by the electoral authority."

scenario "OPERATIONAL_TRANSPARENCY" -- @ref Kiniry/Zimmerman
description
  "A report containing every manifest change, every data or system \
 \ invariant violation, every control procedure violation, and every \
 \ physical security violation shall be prepared and made public by \
 \ the electoral authority after every election."


end

scenario_chart RELIABILITY_REQUIREMENTS
indexing
  partof: "SYSTEM_OPERATIONAL_REQUIREMENTS"
explanation
  "General reliability requirements for any internet election system."

scenario
  "GENERAL_MTBF" -- @ref David Jefferson
description
  "The entire voting service (server side) must have a proven MTBF of \
 \ >168 hours (1 week) under peak expected voting loads the entire \
 \ time."

scenario
  "LIVE_ELECTION_MTBF" -- @ref David Jefferson
description
  "MTBF validation must be demonstrated in multiple tests of \
 \ actual mock elections."

scenario
  "MTBF_CONTRA_DDOS" -- @ref David Jefferson
description
  "MTBF requirements apply only during normal peak operation, not \
 \ during attacks (e.g., DDoS)."

scenario
  "SYSTEM_RECOVERY_TIME" -- @ref David Jefferson
description
  "If service goes down for any reason other than regional natural \
```

```
  \ disaster or malicious attack, service must be restored in no more \
  \ than 10 minutes."

scenario
  "UPTIME" -- @ref David Jefferson
description
  "The system must have three nines (99.9%) uptime."

scenario
  "FAILURE_VALIDATION" -- @ref David Jefferson
description
  "Uptime must be demonstrated by failures in actual mock election \
  \ situations, e.g. tested by sudden loss of power to any server."

scenario
  "MIRRORED_FAILOVER_SERVICE" -- @ref David Jefferson
description
  "The system must have a warm spare in a second data center that can take \
  \ over in case of major failure."

scenario
  "FAILOVER_STAFFING" -- @ref David Jefferson
description
  "The system must be staffed at all times to guarantee the 10 minute \
  \ recovery time."

scenario
  "OPERATION_UNDER_DDOS" -- @ref David Jefferson
description
  "In a federal election the voting system must remain available even \
  \ during a large distributed denial of service attack.  It must be \
  \ able to continue correct operation during a sustained DDoS attack \
  \ on any combination of server side IP addresses (whether at the \
  \ primary server data center or its ISP) at a total level of 100 Gb/s \
  \ with no more than 15s degradation of response time to voters during \
  \ the attack."

scenario
  "DDOS_REFRESH_PERIODICITY" -- @ref David Jefferson
description
  "The DDoS threshold (initially 100 Gb/s) should be evaluated every \
  \ election cycle to see if it has to be raised due to newer \
  \ DDoS attack technologies."

scenario
  "DDOS_ATTACK_VALIDATION" -- @ref David Jefferson
description
  "The ability to survive a DDoS attack must be actually demonstrated \
  \ in the actual network configuration to be used prior to each \
  \ federal election."

scenario
  "DDOS_LOCAL_ELECTION" -- @ref David Jefferson
description
  "Reduced DDoS defense requirements might be acceptable for \
  \ non-federal elections."

end
```

```
scenario_chart SECURITY_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS";
explanation
  "General security requirements for digital elections systems."

-- @ref Rec(2004)11 Security, I. General requirements
scenario "NO_DATA_LOSS" -- @ref Rec(2004)11 Appendix III, D. I. 77.
description
  "Technical and organizational measures shall be taken to ensure that \
 \ no data will be permanently lost in the event of a breakdown or a \
 \ fault affecting the e-voting system."

scenario "VOTER_PRIVACY_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. I. 78.
description
  "The e-voting system shall maintain the privacy of \
 \ individuals.  Confidentiality of voters' registers stored in or \
 \ communicated by the e-voting system shall be maintained."

scenario "SYSTEM_SELF_CHECKS" -- @ref Rec(2004)11 Appendix III, D. I. 79.
description
  "The e-voting system shall perform regular checks to ensure that its \
 \ components operate in accordance with its technical specifications \
 \ and that its services are available."

scenario "SYSTEM_ACCESS_CONTROL" -- @ref Rec(2004)11 Appendix III, D. I. 80.
description
  "The e-voting system shall restrict access to its services, \
 \ depending on the user identity or the user role, to those services \
 \ explicitly assigned to this user or role.  User authentication shall \
 \ be effective before any action can be carried out."

scenario "DATA_PROTECTION" -- @ref Rec(2004)11 Appendix III, D. I. 81.
description
  "The e-voting system shall protect authentication data so that \
 \ unauthorized entities cannot misuse, intercept, modify, or otherwise \
 \ gain knowledge of any of this data.  In uncontrolled \
 \ environments, authentication based on cryptographic mechanisms is \
 \ advisable."

scenario "UNIQUE_IDENTIFICATION" -- @ref Rec(2004)11 Appendix III, D. I. 82.
description
  "Identification of voters and candidates in a way that they can \
 \ unmistakably be distinguished from other persons (unique \
 \ identification) shall be ensured."

scenario "OBSERVATION_DATA" -- @ref Rec(2004)11 Appendix III, D. I. 83.
description
  "E-voting systems shall generate reliable and sufficiently detailed \
 \ observation data so that election observation can be carried \
 \ out.  The time at which an event generated observation data shall be \
 \ reliably determinable.  The authenticity, availability and \
 \ integrity of the data shall be maintained."

scenario "TIME_SYNCHRONIZATION" -- @ref Rec(2004)11 Appendix III, D. I. 84.
description
  "The e-voting system shall maintain reliable synchronized time \
 \ sources.  The accuracy of the time sources shall be sufficient to \
 \ maintain time marks for audit trails and observations data, as well \
```

```
 \ as for maintaining the time limits for registration, nomination, \
 \ voting, or counting."

scenario "SECURITY_COMPLIANCE_RESPONSIBILITY" -- @ref Rec(2004)11 Appendix III, D. I.
    85.
description
  "The electoral authority has overall responsibility for compliance \
 \ with these security requirements, and such compliance shall be assessed by \
 \ independent bodies."

-- @ref Rec(2004)11 Security, II. Requirements in pre-voting stages
scenario "LISTS_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. II. 86.
description
  "The authenticity, availability and integrity of the voters' \
 \ registers and lists of candidates shall be maintained.  The source of \
 \ the data shall be authenticated.  Provisions on data protection shall \
 \ be respected."

scenario "CANDIDATE_PROCESS_TIME_PROVENANCE" -- @ref Rec(2004)11 Appendix III, D. II.
    87.
description
  "The fact that candidate nomination and, if required, the decision \
 \ of the candidate and/or the electoral authority to accept a \
 \ nomination has happened within the prescribed time limits shall be \
 \ ascertainable."

scenario "VOTER_PROCESS_TIME_PROVENANCE" -- @ref Rec(2004)11 Appendix III, D. II. 88.
description
  "The fact that voter registration has happened within the prescribed \
 \ time limits shall be ascertainable."

-- @ref Rec(2004)11 Security, III. Requirements in the voting stage
scenario "ELECTION_DATA_INTEGRITY_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. III
    . 89.
description
  "The integrity of data communicated from the pre-voting stage \
 \ (e.g., voters' registers and lists of candidates) shall be \
 \ maintained.  Data-origin authentication shall be carried out."

scenario "BALLOT_AUTHENTICITY_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. III.
    90.
description
  "It shall be ensured that the e-voting system presents an authentic \
 \ ballot to the voter.  In the case of remote e-voting, the voter shall \
 \ be informed about the means to verify that a connection to the \
 \ official server has been established and that the authentic ballot \
 \ has been presented."

scenario "CAST_VOTE_TIME_PROVENANCE" -- @ref Rec(2004)11 Appendix III, D. III. 91.
description
  "The fact that a vote has been cast within the prescribed time \
 \ limits shall be ascertainable."

scenario "CONTROLLED_SYSTEMS_AND_VOTE_INTEGRITY" -- @design derived from Rec(2004)11
    Appendix III, D. III. 92.
description
  "Election equipment under the control of the electoral authority \
 \ shall be protected against influence that could modify the vote."
```

```
scenario "UNCONTROLLED_SYSTEMS_AND_VOTE_INTEGRITY" -- @ref Kiniry/Zimmerman
description
  "The integrity of the vote must not depend on the security of election \
 \ equipment not under the control of the electoral authority."

scenario "NO_BREADCRUMBS" -- @ref Rec(2004)11 Appendix III, D. III. 93.
description
  "Residual information holding the voter's decision or the display of \
 \ the voter's choice shall be destroyed after the vote has been \
 \ cast.  In the case of remote e-voting, the voter shall be provided \
 \ with information on how to delete, where that is possible, traces \
 \ of the vote from the device used to cast the vote."

scenario "ELIGIBILITY_IMPLIES_VOTE_VOTER_INVARIANTS" -- @ref Rec(2004)11 Appendix III,
    D. III. 94.
description
  "The e-voting system shall at first ensure that a user who tries to \
 \ vote is eligible to vote.  The e-voting system shall authenticate \
 \ the voter and shall ensure that only the appropriate number of votes \
 \ per voter is cast and stored in the electronic ballot box."

scenario "VOTE_CHOICE_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. III. 95.
description
  "The e-voting system shall ensure that the voter's choice is \
 \ accurately represented in the vote and that the sealed vote enters \
 \ the electronic ballot box."

scenario "END_OF_VOTE_PERIOD_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. III. 96.
description
  "After the end of the e-voting period, no voter shall be allowed to \
 \ gain access to the e-voting system.  However, the acceptance of \
 \ electronic votes into the electronic ballot box shall remain open \
 \ for a sufficient period of time to allow for any delays in the \
 \ passing of messages over the e-voting channel."

-- @ref Rec(2004)11 Security, IV. Requirements in post-voting stages
scenario "DATA_COMMUNICATION_INTEGRITY_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D
    . IV. 97.
description
  "The integrity of data communicated during the voting stage \
 \ (e.g. votes, voters' registers, lists of candidates) shall be \
 \ maintained.  Data-origin authentication shall be carried out."

scenario "TALLY_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. IV. 98.
description
  "The counting process shall accurately count the votes.  The counting \
 \ of votes shall be reproducible."

scenario "BALLOT_BOX_AND_TALLY_INVARIANTS" -- @ref Rec(2004)11 Appendix III, D. IV.
    99.
description
  "The e-voting system shall maintain the availability and integrity \
 \ of the electronic ballot box and the output of the counting process \
 \ as long as required."

scenario "ADVERSARY_RESOURCES" -- @ref Kiniry/Zimmerman
description
  "The e-voting system shall be designed and tested with the assumption \
 \ that an adversary has a budget of $10 per voter per election, which they \
```

```
 \ can apply toward any critical subset of votes/voters of their choosing."
end

scenario_chart E2EVIV_SECURITY_REQUIREMENTS
indexing
  partof: "SECURITY_REQUIREMENTS";
  author: "David Jefferson <d_jefferson@yahoo.com>";
  created: "22 June 2014";
  reviewer: "Joe Kiniry <kiniry@galois.com>";
  reviewed: "16 July 2014"
explanation
  "General security requirements for end-to-end verifiable internet \
 \ election systems."

 -- These are requirements for embedding an E2EV system in an Internet
 -- voting environment.  They are over and above the requirements for
 -- the core E2EV itself.  We do not consider usability or accessibility
 -- requirements here.  Some of these requirements will make
 -- accessibility and usability more difficult to achieve.  Still, these
 -- are requirements, and if they cannot be met, or cannot be met
 -- simultaneously with usability and accessibility requirement, then we
 -- have to recommend not implementing an E2EV Internet voting system.

scenario
  "NATIONAL_SECURITY" -- @ref David Jefferson
description
  "If used in federal elections, an Internet voting system is also a \
 \ national security system, and thus must be subject to the highest \
 \ security requirements."

scenario
  "FEDERAL_REQUIREMENTS" -- @ref David Jefferson
description
  "Any Internet voting system used in a public primary or general \
 \ election in the U.S. for federal or state legislative, executive, \
 \ or judicial office, or recall election, or statewide initiative or \
 \ referendum, must meet all of the requirements in this document."

scenario
  "LOCAL_REQUIREMENTS" -- @ref David Jefferson
description
  "Reduced security requirements might be appropriate for county, \
 \ municipal, or other kinds of elections"

scenario
  "AUTOMATED_REGISTRATION_FRAUD" -- @ref David Jefferson
description
  "Automated registration fraud must not be possible."
 -- Eligibility & Registration (online registration, automated
 -- registration fraud, and change of credentials): DJ doesn't know yet
 -- what to write here regarding requirements.  But obviously any
 -- automated registration fraud can be used to affect the outcome of
 -- elections.

scenario
  "CLIENT_SIDE_AUTHENTICITY" -- @ref David Jefferson
description
  "There must be a means by which any third party can determine if the \
 \ client-side software is genuine."
```

```
-- Authentication of service: Not sure what requirement should be
-- here.  The intent is to somehow ascertain that the E2EV software
-- on the client-side is genuine.  Presumably that E2EV software will
-- authenticate the remote server.

scenario
  "AUTHENTICATION_INDEPENDENCE" -- @ref David Jefferson
description
  "The security of authentication must not be affected by \
 \ any potential breach of any public or commercial databases."

scenario
  "ZERO_KNOWLEDGE_AUTHENTICATION" -- @ref David Jefferson
description
  "It should not be possible for an attacker to impersonate voters \
 \ even if the entire server database used for authentication is \
 \ compromised."

scenario
  "AUTHENTICATION_CREDENTIAL_REESTABLISHMENT" -- @ref David Jefferson
description
  "In some cases of security breach it must be possible to require all \
 \ voters in a jurisdiction to re-establish credentials."
end

scenario_chart PRIVACY_REQUIREMENTS
indexing
  partof: "SECURITY_REQUIREMENTS"
explanation
  "General privacy requirements for end-to-end verifiable internet \
 \ election systems."
-- violations of vote privacy are not generally detectable
-- violations of vote privacy are irreversible
-- violations of vote privacy enable vote coercion and vote selling
-- vote privacy cannot be verified by testing; it can only be ascertained by expert
    analysis of architecture and code

scenario
  "E2E_VOTE_PRIVACY" -- @ref David Jefferson
description
  "Vote privacy must be preserved end-to-end insofar as mathematically \
 \ possible."

scenario
  "VOTE_PRIVACY_INVIOLATE" -- @ref David Jefferson
description
  "Vote privacy cannot be waived by voters."

scenario
  "MALWARE_PRESENCE" -- @ref David Jefferson
description
  "Vote privacy must not be violated even in the presence of arbitrary \
 \ malicious code on the client platform, including phony client \
 \ software, malicious client wrappers, MITM code between the user and \
 \ the E2EV interface, malicious browser plugins or scripts, \
 \ keyloggers, etc."
 -- This requirement will seriously complicate the user interface an
 -- usability of the system, but is absolutely essential.
```

```
scenario
  "REMOTE_MONITORING" -- @ref David Jefferson
description
  "Voting should not be permitted from client platforms known to have \
 \ remote monitoring software installed that could be used to monitor \
 \ or log voting activity and that cannot be turned off by the voter. \
 \ (All mobile platforms had, and probably still do have, such remote \
 \ monitoring software.)"

scenario
  "CLIENT_SIDE_CHANNELS" -- @ref David Jefferson
description
  "The client software of the voting system must not send data to any \
 \ IP address except those associated with the vote server and the \
 \ basic infrastructure servers of the Internet."

scenario
  "SOCIAL_MEDIA_SIDE_CHANNELS" -- @ref David Jefferson
description
  "The client should not provide any information to third parties, \
 \ e.g., Facebook, Twitter, etc. regarding the act of voting."

scenario
  "NO_TRACKING" -- @ref David Jefferson
description
  "There must be no tracking devices or tracking logic in the vote \
 \ client."

scenario
  "NO_BREADCRUMBS_DETAILS" -- @ref David Jefferson
description
  "The client software must leave no files or other persistent data on \
 \ the platform regarding the vote transaction but for an optional \
 \ file containing information needed for subsequent verification that \
 \ the voter's ballot is included in the election canvass: no cookies \
 \ or other session files, no temporary files."

scenario
  "TRANSIENT_DATA_CLEANUP" -- @ref David Jefferson
description
  "The client software should explicitly erase (i.e., overwrite) all \
 \ transient copies of vote-transaction data, e.g. data in registers, \
 \ caches, RAM, and virtual memory."

scenario
  "FORENSICALLY_SECURE" -- @ref David Jefferson
description
  "It should not be possible even for client-side forensic tools to \
 \ retrieve any information regarding the voting transaction after the \
 \ voting session is ended."

scenario
  "REMOTE_ADMINISTRATION_FORBIDDEN" -- @ref David Jefferson
description
  "The voting system should not support platforms that have remote \
 \ administration or remote control tools installed that cannot be \
 \ turned off by the voter."

scenario
```

```
    "INVULNERABLE_TO_ELECTION_MALWARE" -- @ref David Jefferson
description
  "The voting system must not be vulnerable to malware designed to \
 \ modify votes before they are input to the E2EV system."
 -- This will seriously complicate the human interface and usability
 -- of the voting system, but is absolutely essential.  Malware can be
 -- in many forms: completely phony or "alternative" client app,
 -- client wrapper, client-side MITM, browser plugin, client APT, etc.

scenario
  "CLIENT_SYSTEM_AUTHENTICATION" -- @ref David Jefferson
description
  "The voting system server must authenticate that it is communicating \
 \ with a genuine vote client during a voting session."
 -- This will complicate, but not eliminate, the possibility of
 -- client-side malware.  @see CLIENT_SIDE_AUTHENTICITY.

scenario
  "PENETRATION_ATTACKS" -- @ref David Jefferson
description
  "Deny penetration attacks. (DJ doesn't know what to write about \
 \ this.)"

scenario
  "APT_ATTACKS" -- @ref David Jefferson
description
  "Deny advanced persistent threat attacks.  (DJ doesn't know what to \
 \ write about this.)"

scenario
  "INSIDER_ATTACKS" -- @ref David Jefferson
description
  "Something about insider attacks being impossible.  (DJ doesn't know \
 \ what to write about this.)"

scenario
  "COERCION_PREVENTION" -- @ref David Jefferson
description
  "There must be no way for voters to prove to another party any \
 \ information regarding how they voted in any race beyond what is \
 \ mathematically deducible from the final distribution of votes."
  -- @see RECEIPT_FREEDOM

scenario
  "SOFTWARE_INDEPENDENCE" -- @ref Ron Rivest
description
  "The system must witness software independence: the tabulation \
 \ record must not rely solely on software."

scenario
  "DIGITAL_EVIDENCE_NOT_A_RECEIPT"
description
  "Digital evidence (e.g., photographing a ballot or video recording \
 \ the casting process) of the voting process must not violate receipt \
 \ freedom."
end

scenario_chart CERTIFICATION_AND_RECERTIFICATION_REQUIREMENTS
indexing
```

```
  partof: "SECURITY_REQUIREMENTS"
explanation
  "General security requirements relating to certification of digital \
 \ elections systems."
end

scenario_chart USABILITY_REQUIREMENTS
indexing
  partof: "TECHNICAL_REQUIREMENTS"
explanation
  "General usability requirements of digital elections systems."

scenario
  "MANDATORY_USABILITY_TESTING" -- @ref Kiniry/Zimmerman
description
  "Usability testing for disabled and abled voters shall be performed, \
 \ and the reports of the testing made public. The system must achieve \
 \ satisfactory usability testing results before being used in a \
 \ binding election."

scenario
  "VOTE_CONFIRMATION" -- @ref David Jefferson
description
  "If a voter receives the final 'Thank you for voting' confirmation, \
 \ then she/he can be certain the ballot was recorded."

scenario
  "UNCERTAIN_VOTER_REVOTE" -- @ref David Jefferson
description
  "If the voter is uncertain about the state of their ballot, he/she \
 \ is free to attempt to vote again."

end
```

Bibliography

[1] *2012 Election Administration and Voting Survey*. 2013.

[2] Claudia Z Acemyan et al. "Usability of Voter Verifiable, End-to-end Voting Systems: Baseline Data for Helios, Prêt à Voter, and Scantegrity II". In: *The USENIX Journal of Election Technology and Systems* (2014), p. 26.

[3] Ben Adida. "Helios: Web-based Open-Audit Voting". In: *USENIX Security*. 2008. URL: https://www.usenix.org/legacy/events/sec08/tech/full_papers/adida/adida.pdf.

[4] Ben Adida et al. "Electing a University President using Open-Audit Voting: Analysis of real-world use of Helios". In: *USENIX EVT/WOTE*. 2009. URL: https://www.usenix.org/legacy/event/evtwote09/tech/full_papers/adida-helios.pdf.

[5] *asm.js: an extraordinarily optimizable, low-level subset of JavaScript*. URL: http://www.asmjs.org/.

[6] American Political Science Association et al. "Findings and recommendations of the special committee on service voting". In: *American Political Science Review* 46.2 (1952), pp. 512–523.

[7] Atlassian. *Comparing Workflows*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows (visited on 04/22/2015).

[8] David Bismark et al. "Experiences Gained from the first Prêt à Voter Implementation". In: *First International Workshop on Requirements Engineering for e-Voting Systems (RE-VOTE)*. IEEE. 2009, pp. 19–28. DOI: 10.1109/RE-VOTE.2009.5.

[9] Bruno Blanchet. "Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif". English. In: *Foundations of Security Analysis and Design VII*. Ed. by Alessandro Aldini, Javier Lopez, and Fabio Martinelli. Vol. 8604. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 54–87. ISBN: 978-3-319-10081-4. DOI: 10.1007/978-3-319-10082-1_3. URL: http://dx.doi.org/10.1007/978-3-319-10082-1_3.

[10] Philippe Bulens, Damien Giry, and Olivier Pereira. "Running mixnet-based elections with Helios". In: *USENIX EVT/WOTE*. 2011. URL: http://www.usenix.org/events/evtwote11/tech/final_files/Bulens.pdf.

[11] Craig Burton et al. "Using Prêt à Voter in Victorian State elections". In: *USENIX EVT/WOTE*. 2012. URL: https://www.usenix.org/system/files/conference/evtwote12/evtwote12-final9_0.pdf.

[12] Michael D Byrne, Kristen K Greene, and Sarah P Everett. "Usability of voting systems: Baseline data for paper, punch cards, and lever machines". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2007, pp. 171–180.

[13] Richard Carback et al. "Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy". In: *USENIX Security*. 2010. URL: https://www.usenix.org/legacy/events/sec10/tech/full_papers/Carback.pdf.

[14] Carter Center. *Internet Voting Pilot: Norway's 2013 Parliamentary Elections*. http://www.cartercenter.org/resources/pdfs/peace/democracy/Carter-Center-Norway-2013-study-mission-report2.pdf. 2014.

[15] *Center for Advanced Software Analysis: Software Tools*. URL: http://casa.au.dk/software-tools/.

[16]   Scott Chacon. *Pro Git*. Berkeley, CA New York, NY: Apress,Distributed to the book trade worldwide by Spring Science+Business Media, 2014. ISBN: 978-1484200773.

[17]   David Chaum, Peter Y. A. Ryan, and Steve Schneider. "A Practical Voter-Verifiable Election Scheme". English. In: *Computer Security – ESORICS 2005*. Ed. by Sabrinade Capitani di Vimercati, Paul Syverson, and Dieter Gollmann. Vol. 3679. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 118–139. ISBN: 978-3-540-28963-0. DOI: 10.1007/11555827_8. URL: http://dx.doi.org/10.1007/11555827_8.

[18]   David Chaum et al. "Accessible voter-verifiability". In: *Cryptologia* 33.3 (2009), pp. 283–291.

[19]   David Chaum et al. "Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes". In: *IEEE Transactions on Information Forensics and Security* 4.4 (2009), pp. 611–627. ISSN: 1556-6013. DOI: 10.1109/TIFS.2009.2034919.

[20]   David Chaum et al. "Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes". In: *USENIX EVT*. 2008. URL: https://www.usenix.org/legacy/event/evt08/tech/full_papers/chaum/chaum.pdf.

[21]   *CloudFlare, Inc.* URL: http://www.cloudflare.com/.

[22]   Alistair Cockburn and Laurie Williams. "The costs and benefits of pair programming". In: *Extreme programming examined* (2000), pp. 223–247.

[23]   *CVK: Crypto Verification Kit*. URL: http://research.microsoft.com/en-us/projects/cvk/.

[24]   Alex Delis et al. *Pressing the Button for European Elections 2014: Public attitudes towards Verifiable E-Voting In Greece*. https://drive.google.com/file/d/0B-mtbRwyPn_SdnpMRzBKcEZWUm8/view?usp=sharing. 2014.

[25]   Aleks Essex et al. "Punchscan in practice: an E2E election case study". In: *Proceedings of Workshop on Trustworthy Elections*. 2007.

[26]   Michael Fagan. "Design and code inspections to reduce errors in program development". In: *Software pioneers*. Springer, 2002, pp. 575–607.

[27]   Python Software Foundation. *doctest — Test interactive Python examples*. 2015. URL: https://docs.python.org/3/library/doctest.html (visited on 05/05/2015).

[28]   Federal Constitutional Court of Germany. *Docket Nos. 2 BvC 3/07 & 2 BvC 4/07*. 2009. URL: https://www.bundesverfassungsgericht.de/SharedDocs/Entscheidungen/EN/2009/03/cs20090303_2bvc000307en.html (visited on 04/20/2015).

[29]   Kristian Gjøsteen. "The Norwegian Internet Voting Protocol". English. In: *E-Voting and Identity*. Ed. by Aggelos Kiayias and Helger Lipmaa. Vol. 7187. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 1–18. ISBN: 978-3-642-32746-9. DOI: 10.1007/978-3-642-32747-6_1. URL: http://dx.doi.org/10.1007/978-3-642-32747-6_1.

[30]   Milos Gligoric et al. "Comparing non-adequate test suites using coverage criteria". In: *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM. 2013, pp. 302–313.

[31]   Rop Gonggrijp et al. "RIES - Rijnland Internet Election System: A Cursory Study of Published Source Code". English. In: *E-Voting and Identity*. Ed. by PeterY.A. Ryan and Berry Schoenmakers. Vol. 5767. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 157–171. ISBN: 978-3-642-04134-1. DOI: 10.1007/978-3-642-04135-8_10. URL: http://dx.doi.org/10.1007/978-3-642-04135-8_10.

[32]   *How to Vote: Wombat Voting System*. http://www.wombat-voting.com/how-to-vote.

[33]   Engelbert Hubbers, Bart Jacobs, and Wolter Pieters. "RIES - Internet Voting in Action". In: *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*. Vol. 1. IEEE. 2005, pp. 417–424.

[34]   Engelbert Hubbers et al. "Description and analysis of the RIES internet voting system". In: *Report of the Eidhoven Institute for the Protection of Systems and Information*. Faculty of Mathematics and Computer Science Eindhoven University of Technology, 2008.

[35] Laura Inozemtseva and Reid Holmes. "Coverage is not strongly correlated with test suite effectiveness". In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 435–445.

[36] *JSCert: Certified JavaScript*. URL: http://www.jscert.org/.

[37] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: http://doi.acm.org/10.1145/357172.357176.

[38] John Launchbury et al. "Application-Scale Secure Multiparty Computation". English. In: *Programming Languages and Systems*. Ed. by Zhong Shao. Vol. 8410. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 8–26. ISBN: 978-3-642-54832-1. DOI: 10.1007/978-3-642-54833-8_2. URL: http://dx.doi.org/10.1007/978-3-642-54833-8_2.

[39] Anil Madhavapeddy et al. "Unikernels: Library Operating Systems for the Cloud". In: *SIGPLAN Notices* 48.4 (Mar. 2013), pp. 461–472. ISSN: 0362-1340. DOI: 10.1145/2499368.2451167. URL: http://doi.acm.org/10.1145/2499368.2451167.

[40] *Mirage OS*. URL: http://www.openmirage.org/.

[41] George C. Necula. "Proof-Carrying Code. Design and Implementation". English. In: *Proof and System-Reliability*. Ed. by Helmut Schwichtenberg and Ralf Steinbrüggen. Vol. 62. NATO Science Series. Springer Netherlands, 2002, pp. 261–288. ISBN: 978-1-4020-0608-1. DOI: 10.1007/978-94-010-0413-8_8. URL: http://dx.doi.org/10.1007/978-94-010-0413-8_8.

[42] United States. General Accounting Office. *Voters with disabilities: access to polling places and alternative voting methods*. US General Accounting Office, 2001.

[43] *OSCE/ODIHR Election Assessment Mission Report*. http://www.osce.org/odihr/elections/netherlands/24322?download=true. 2006.

[44] C Pilato. *Version control with Subversion*. Sebastopol, CA: O'Reilly Media, 2008. ISBN: 0596510330.

[45] Raluca Ada Popa et al. "CryptDB: Protecting Confidentiality with Encrypted Query Processing". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: ACM, 2011, pp. 85–100. ISBN: 978-1-4503-0977-6. DOI: 10.1145/2043556.2043566. URL: http://doi.acm.org/10.1145/2043556.2043566.

[46] Stefan Popoveniuc and Ben Hosp. "An introduction to Punchscan". In: *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*. Robinson College United Kingdom. 2006, pp. 28–30.

[47] Stefan Popoveniuc and Ben Hosp. "An Introduction to PunchScan". English. In: *Towards Trustworthy Elections*. Ed. by David Chaum et al. Vol. 6000. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 242–259. ISBN: 978-3-642-12979-7. DOI: 10.1007/978-3-642-12980-3_15. URL: http://dx.doi.org/10.1007/978-3-642-12980-3_15.

[48] Noel Runyan. "Improving access to voting: A report on the technology for accessible voting systems". In: *Retrieved October* 1 (2007), p. 2008.

[49] P.Y.A. Ryan et al. "Prêt à Voter: a Voter-Verifiable Voting System". In: *Information Forensics and Security, IEEE Transactions on* 4.4 (2009), pp. 662–673. ISSN: 1556-6013. DOI: 10.1109/TIFS.2009.2033233.

[50] *Security Review: Helios Online Voting*. https://cubist.cs.washington.edu/Security/2009/03/13/security-review-helios-online-voting/. 2009.

[51] Claire M. Smith. *Convenience Voting and Technology: The Case of Military and Overseas Voters (Elections, Voting, Technology)*. Palgrave Macmillan, 2014. ISBN: 1137398582. URL: http://www.amazon.com/Convenience-Voting-Technology-Military-Elections/dp/1137398582\%3FSubscriptionId\%3D0JYN1NVW651KCA56C102\%26tag\%3Dtechkie-20\%26linkCode\%3Dxm2\%26camp\%3D2025\%26creative\%3D165953\%26creativeASIN\%3D1137398582.

[52] Claire M Smith. "Time to Move: Overseas and Military Voter State Policy Innovation". In: (2011).

[53] Wouter Swierstra. "Xmonad in Coq (experience report): Programming a window manager in a proof assistant". In: *ACM SIGPLAN Notices*. Vol. 47. 12. ACM. 2012, pp. 131–136.

[54] *The Haskell Lightweight Virtual Machine (HaLVM)*. URL: https://galois.com/project/halvm/.

[55] Ken Thompson. "Reflections on Trusting Trust". In: *Communications of the ACM* 27.8 (Aug. 1984), pp. 761–763. ISSN: 0001-0782. DOI: 10.1145/358198.358210. URL: http://doi.acm.org/10.1145/358198.358210.

[56] James E Tomayko. "A comparison of pair programming to inspections for software defect reduction". In: *Computer Science Education* 12.3 (2002), pp. 213–222.

[57] Georgios Tsoukalas et al. "From Helios to Zeus". In: *USENIX EVT/WOTE*. 2013. URL: https://www.usenix.org/conference/evtwote13/workshop-program/presentation/Tsoukalas.

[58] U.S. Election Assistance Commission. *UOCAVA Pilot Program Testing Requirements—August 25, 2010*. 2010. URL: http://www.eac.gov/assets/1/Documents/UOCAVA_Pilot_Program_Testing\%20Requirements\%20August\%208\%202010.pdf.

[59] Kim Walden. *Seamless object-oriented software architecture : analysis and design of reliable systems*. New York: Prentice Hall, 1995. ISBN: 0130313033.

[60] David A. Wheeler. "Fully Countering Trusting Trust through Diverse Double Compilation". PhD thesis. George Mason University, 2009. URL: http://www.dwheeler.com/trusting-trust/.

[61] *Xen Project*. URL: http://www.xenproject.org/.

[62] Xuejun Yang et al. "Finding and understanding bugs in C compilers". In: *ACM SIGPLAN Notices*. Vol. 46. 6. ACM. 2011, pp. 283–294.

[63] Shin Yoo and Mark Harman. "Regression testing minimization, selection and prioritization: a survey". In: *Software Testing, Verification and Reliability* 22.2 (2012), pp. 67–120.

[64] Filip Zagórski et al. "Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System". English. In: *Applied Cryptography and Network Security*. Ed. by Michael Jacobson et al. Vol. 7954. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 441–457. ISBN: 978-3-642-38979-5. DOI: 10.1007/978-3-642-38980-1_28. URL: http://dx.doi.org/10.1007/978-3-642-38980-1_28.

[65] Philip Zimmermann. *The Official PGP User's Guide*. Cambridge, MA: The MIT Press, 1995. ISBN: 0-262-74017-6.