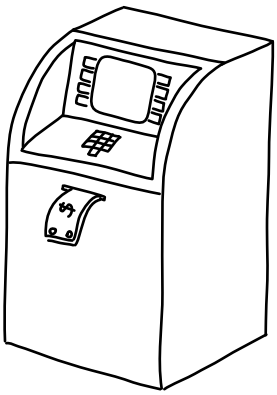
1 Voting

What makes the voting problem so hard?

When I say the "voting problem," I'm referring specifically to the system that collects and counts votes. There are many other parts of the election process that I'm not going to address in this dissertation, such as voter registration, electoral systems, and election campaigning. The collection and counting of votes has been particularly controversial in the United States due to problems with electronic voting in recent elections.

One of the great things about doing election-related research is that just about everyone immediately understands why it's important. In my experience, whenever elections are the topic of conversation, people have a lot to say about their opinions on the matter. It's encouraging to see that so many people care deeply about democracy.

In conversations about the voting problem, there seem to be four ideas in particular that come up all the time. It's not unusual to think that running a fair election ought to be a straightforward task—after all, in some sense, it's just counting. To give you a taste of why the voting problem is not as easy as it might seem, let's begin by examining these four suggestions.

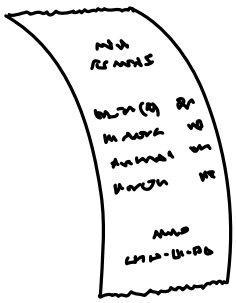


Banking machines work fine, so voting machines should be no problem. On the surface, banking machines and voting machines seem similar: users walk up and make selections on a touchscreen to carry out a transaction. One of the largest vendors, Diebold Inc., even produces both kinds of machines. But the incentives and risks are very different.

Banking machines have money inside—the bank's money. If money goes missing, you can bet the bank will find out right away and be strongly motivated to fix the problem. If the bank machine incorrectly gives out too much cash, the bank loses money; if it gives out too little, the bank will be dealing with irate customers. Everything about the bank transaction is recorded, from the entries in your bank statement to the video recorded by the camera in most bank machines. That's because

the bank has a strong incentive to audit that money and track where it goes. If the machine makes mistakes, the bank loses—either they expend time and money correcting your problem, or you will probably leave and take your business to another bank.

With voting machines, it's another story altogether. Voting machines aren't supposed to record video or keep any record that associates you with your votes, because your ballot is supposed to be secret. You don't receive any tangible confirmation that your vote was counted, so you can't find out if there's a problem. Anybody can stand to gain by causing votes to be miscounted—a voter, pollworker, election administrator, or voting machine programmer—and the consequences are much harder to reverse. Correcting an error in your bank balance is straightforward, but the only way to fix an improperly counted election is to do an expensive manual recount or run the whole election again. And if you're unhappy with the way your vote was handled, you can't easily choose to vote on a competitor's machine.



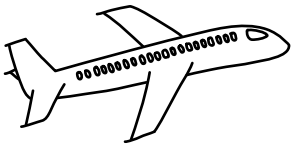
Give each voter a printed receipt, just like we do for any other transaction. The surface comparison between voting and a financial transaction also leads many people to suggest that receipts are the answer. But the purpose of a receipt is quite different from what is needed to ensure an accurate election.

When you buy something, the receipt confirms that you paid for it. If there turns out to be a problem with the product, you can use the receipt to get your money back or to get the defective product exchanged.

When talking about a receipt from a voting machine, what most people have in mind is a printed record of the choices you made, just like a receipt from a cash register. If you took home such a receipt, what would you do with it? There's nothing to refund, and you can't use a receipt to get an exchange on a defective politician. The receipt could record the choices you made, but the receipt alone doesn't assure that those choices were counted in the final result. In fact, if the receipt constitutes proof of which choices you made, it can be sold—

defeating the whole point of the secret ballot, which is to avoid the corruption that vote-buying campaigns can cause.

A truly useful voting "receipt" would do exactly the opposite: it would *not* reveal which choices you made but *would* let you confirm that your choices were counted. Although these two requirements sound paradoxical, researchers have invented a variety of schemes that achieve them through the clever use of cryptography. However, a key weakness of the schemes proposed so far is that they rely on advanced mathematics, with a counting process that would be a mystery to all but a tiny minority of voters. This would run counter to the democratic principle of transparent elections. Researchers are continuing to search for simpler verification schemes that can be understood by an acceptably large fraction of the public.

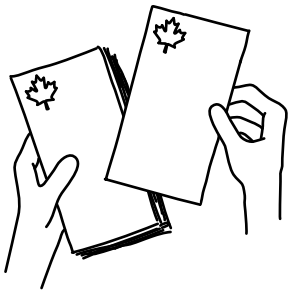


If we can trust computers to fly airplanes, we can trust computers to run elections. The comparison between airplanes and elections misses at least three key differences.

First, the visibility of failure is different. An airplane cannot secretly fail to fly. When an airplane crashes, it makes headlines; everybody knows. A forensic investigation takes place, and if the crash is due to a manufacturing defect, the airplane manufacturer may be sued for millions of dollars. But an election system can produce incorrect results without any obvious signs of failure. Therefore, we require something more from election system software than what we require from airplane software. A successful election system must not only *work* correctly; it must also allow the public to *verify* that it worked correctly.

Second, the target audience is different. Commercial airplanes are designed to be flown by pilots with expert training, but voting machines have to be set up by pollworkers and operated by the general public. Our trust in airplanes is a combination of trust in the equipment and trust in the pilots who operate it. Whereas pilots have to log hundreds of hours of flight time to get a license, pollworkers are often hired on a temporary basis with only an afternoon or a day of training.

Third, security violations affect the perpetrators differently. Pilots and flight attendants are strongly motivated to uphold security procedures because their own lives could be at risk. A rogue voter or pollworker, on the other hand, would have more to gain and less to lose by surreptitiously changing the outcome of an election.



Count the ballots by hand—it works for the Canadians. Ballots are considerably longer and more complicated in the United States than in many other countries. Whereas there is just one contest in a Canadian federal election (each voter selects a Member of Parliament), ballots in the United States can contain dozens of contests. For example, a typical ballot[1] for the November 2004 general election in Orange County, California contained 7 offices and 16 referenda, for a total of 23 contests that would have to be tallied by hand. Ballots in Chicago, Illinois that year[2] were even longer: ten pages of selections, consisting of 15 elected offices, confirmations of 74 sitting judges, and one referendum—a total of 90 contests. When you appreciate the scale of the task, it becomes easier to understand why many people are motivated to automate the process with computers. Hand-counting paper ballots is by no means impossible, but it would be considerably more expensive and time-consuming in the United States than in other countries with simpler ballots.

* * *

In summary, voting is especially challenging because:

- All involved parties can gain by corrupting an election.
- Results can be incorrect without an obvious failure.
- Democracy demands verifiability, not just correctness.
- Voter privacy and election transparency are in conflict.
- Elections must be accessible and usable by the public.
- Ballots in the United States are long and complex.

[1] The example here is Orange County's ballot type SB019 from November 2004, available in NIST's collection of sample ballots at `http://vote.nist.gov/ballots.htm`.

[2] This refers to the "Code 9" ballot style in Cook County, Illinois (also available in NIST's collection), used in Ward 19, Precincts 28, 43(R), 48, 50(R), and 66, as well as precincts in Wards 21 and 34.

How does an election work?

Running an election is a tremendous organizational task. In the end, it does come down to counting, but it's what's being counted that makes it such a challenge. Election administrators are, in effect, trying to take a fair and accurate measurement of the preferences of the entire population—a controlled experiment on a grand scale. As any psychologist will tell you, performing experimental measurements on human subjects is fraught with logistic pitfalls and sources of error. But elections are worse: virtually everybody has an incentive to actively bias the measurement toward their own preferred outcome. Thus, elections involve a security element as well, unlike most scientific measurements.

As if that weren't enough, a typical election in the United States is not just one opinion poll but many different polls conducted on the same day—for federal, state, and local elected offices, as well as state and local referenda—and each poll has to be localized to a specific region. Each contest appears on some ballots but not others, resulting in different combinations of contests on different ballots. Each combination is called a *ballot style*. Because there are so many kinds of districts (such as congressional districts, state assembly districts, municipalities, hospital districts, and school districts), and district boundaries of each kind often run through districts of other kinds, there can be over a hundred different ballot styles in a single county. There can also be multiple ballot styles at one polling place, if it serves voters on both sides of a district boundary, or if there are different ballots for voters of different political parties.

Process. Here is a simplified breakdown of the election process, setting aside voter registration and considering only the collection and counting of votes. The events before, during, and after actual voting make up the three stages of the process: *preparation*, *polling*, and *counting*.

- *Preparation.* Before any votes can be cast, election officials must prepare the ballots. Election officials map out all the different kinds of political districts, assemble the contests that are relevant to each political district, compose the contests into ballot styles, and determine which ballot styles go to which polling places.
- *Polling.* At polling places, pollworkers sign in each voter and make sure that each voter gets the correct style of ballot. Each voter makes their selections privately and casts a ballot. Voters may also have the option of voting by mail or participating in "early voting" by showing up in person at a special polling place before election day.
- *Counting.* The records of cast votes are counted, either at the polling places or at a central election office. If counting initially occurs at polling places, the counts are then transmitted to the central office for tallying. The votes for each contest are extracted from all the ballots on which that contest appears, and tallied to produce a result.

Equipment. The preceding description is intentionally ambiguous about whether paper or electronic voting is used, because the same three stages take place regardless of the type of equipment.

If paper ballots are used, a layout is prepared for each ballot style, usually designed on a computer. Election administrators estimate how many ballots of each style will be needed so that an adequate number can be printed for distribution to polling places. After being marked, paper ballots can be counted by hand or scanned on machines (called *optical scanning* machines). The scanning can take place at the polls (*precinct count optical scanning*), where each voter feeds their ballot through a scanning machine into a ballot box, or it can take place at a central office, where all the paper ballots are gathered and scanned in high-speed machines after polls close (*central count optical scanning*).

An alternative to paper ballots is to make selections on an electronic voting machine that directly records the selections in

computer memory. These machines are called *direct recording electronic* (DRE) machines. In this case, preparing ballots consists of producing *ballot definition* files on electronic media (such as memory cards or cartridges) to be placed in voting machines. The ballot definition determines what will be displayed to the voter. (Machines for scanning paper ballots also require ballot definitions that specify how the marks on the paper should be counted.) Some DRE machines also print a *voter-verified paper audit trail* (VVPAT)—a paper record of the voter's selections that is shown to the voter for confirmation, but kept sealed inside the machine to enable later recounts.

* * *

To sum up, there are three broad categories of elections in terms of how machines are used:

1. Vote on paper; count by hand.
2. Vote on paper; count by machine.
3. Vote on machine; count by machine.

(3a. The voting machine may also produce a paper record.)

Why use computers for elections?

As the preceding description makes clear, all three stages of the election process involve complex and detail-oriented work. Preparation involves managing information about all the different contests, candidates, and ballot styles. Polling involves distributing this information and collecting results from all the polling places. Counting involves consolidating all the votes for each candidate in each contest across all the ballots and ballot styles. With so many contests on the ballot, computers can make this process much easier.

It's not surprising that election administrators have looked to computers for help with elections. Computers are used to great benefit in automating a broad range of complex and repetitive tasks and for recordkeeping functions throughout all kinds of government agencies. Running an election involves organizing and processing a lot of information, such as ballot descriptions and vote tallies, and databases are effective tools for managing this information.

The appeal of computers goes beyond their potential to increase the speed and accuracy of the count. Computerized vote-entry machines have much greater flexibility than paper ballots in the method of presenting contests and choices to voters. They can walk voters through the voting process, provide more detailed instructions, and prevent overvotes. They eliminate the possibility of ambiguous or improperly scanned marks on paper. They can offer a larger selection of languages. They can point out contests that a voter may have missed before finalizing the marked ballot. They can even read the names of candidates aloud, in headphones, for voters who have trouble reading or voters who are blind. Some voters have physical disabilities that prevent them from using pencil and paper. Computerized vote-entry machines allow people to vote using a variety of input devices, such as large buttons, foot pedals, head-controlled switches, or switches controlled by air pressure ("sip-and-puff" devices).

All of these things become possible when the voting process is conducted by an interactive computer program instead of an inert piece of paper. There appears to be a substantial rate of voter errors when voting on paper ballots—in a Rice University study of paper ballots [24], over 11% of the 126 ballots collected contained at least one error. A friendlier and richer voting interface offered by a computer might help voters avoid making mistakes. Furthermore, the principle of equal rights demands that we provide a way for disabled citizens to cast their votes privately and independently.

* * *

In short, computers can offer several advantages:

- Computers can help manage election-related data.
- Computers can count and tally votes faster.
- Counting by computer avoids human counting errors.
- Computers can offer a richer user interface to voters, potentially improving accessibility and voter accuracy.

Depending on how computers are used in an election, some or all of these advantages may apply.

For this type of election:	Computers could be used to:			
	manage election data	speed up counting	reduce counting error	enrich voting user interface
1. Vote on paper; count by hand.	●			
2. Vote on paper; count by machine.	●	●	●	
3. Vote on machine; count by machine.	●	●	●	●

Figure 1.1. Advantages that computers could *potentially* offer for elections.

How did electronic voting become controversial?

In November 2000, Florida's confusing "butterfly ballot" and heavily disputed punch-card recounts [2, 85] brought highly public embarrassment to the United States election system. The election system suffered widespread criticism on many fronts, particularly for using an outdated counting mechanism. Determined to avoid repeating this fiasco, policymakers and election administrators looked to new technology for a solution. The result was a growing wave of interest in electronic voting, which many hoped would eliminate the ambiguity of punch cards and provide fast, accurate counts.

Two years later, the U. S. Congress passed the Help America Vote Act (HAVA) [78], authorizing hundreds of millions of dollars to be spent on new voting machines. Disability organizations were optimistic about the new requirement for "at least one direct recording electronic voting system or other voting system equipped for individuals with disabilities at each polling place." But computer scientists warned against a hasty switch to electronic voting, citing damage to the transparency and reliability of elections. Though electronic voting machines were already in use in some localities (more than 10% of registered voters used them in 2000 [22]), their adoption surged after HAVA passed in 2002.

In early 2003, election activist Bev Harris made a startling discovery [32]. She used Google to search for "Global Election Systems"—the old name of the company that was acquired by Diebold and renamed "Diebold Election Systems." Diebold Election Systems is one of the heavyweights of the United States election systems industry; its touchscreen voting machine, the AccuVote-TS, was the leading DRE machine used in the 2004 United States election [22]. By following the links from her search results, Harris found a completely unprotected Internet site containing a large collection of company files, including the source code for the AccuVote-TS.

Researchers at Johns Hopkins University and Rice University examined this source code and published a landmark report [43] in May 2004, detailing their discovery of "significant and wide-reaching security vulnerabilities." They discovered that voters could vote multiple times and perform administrative functions; they found that cryptography was both misused and missing where it should have been used; and they expressed a lack of confidence in the quality of the software in general, concluding that it was "far below even the most minimal security standards applicable in other contexts." Their findings starkly contradicted Diebold's public claims that its system was "state-of-the-art," "reliable," "accurate," and "secure" [20].

The state of Maryland then commissioned reviews of the same system from two other agencies: Science Applications International Corporation (SAIC) and RABA Technologies. The SAIC report [72], released in September 2003, confirmed that the system was "at high risk of compromise," and the RABA report [64], released in January 2004, agreed that the "general lack of security awareness, as reflected in the Diebold code, is a valid and troubling revelation."

In the 2004 U. S. general election, over 30% of voters cast their votes on electronic voting machines [22]. Voters called in thousands of reports of machine problems, including total breakdowns, incorrectly displayed ballots, premarked choices on the ballot, incorrectly recorded votes, undesired cancellation of ballots or selections, and nonfunctioning or incorrect audio [82].

Since 2004, further investigations have continued to tear down the façade of confidence in the security of voting machines, the claims of vendors, and the testing regime under which the machines were certified. Media story after media story reported on conflicts of interest, regulatory failures, and newly exposed technical vulnerabilities in all the major voting systems, not just Diebold's.

In the summer of 2007, the California Secretary of State conducted a "top-to-bottom review" of the voting systems used

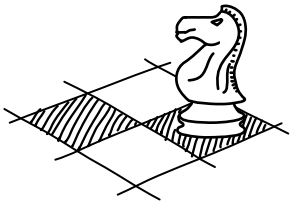
in California, in which I had the opportunity to participate as a reviewer. This was the broadest review of voting system source code to date; the review included source code for DRE machines and optical scan machines from each of three major vendors (Diebold Election Systems, Sequoia Voting Systems, and Hart InterCivic), as well as the election management software responsible for ballot preparation and tallying. However, the review teams only had five weeks to examine the source code. Despite the short time frame, they found serious and pervasive security problems in every system reviewed [7, 12, 35]. The software was not written defensively; security measures were inadequate, misapplied, or poorly implemented; the presence of numerous elementary mistakes suggested that thorough testing had not been done. In particular, every system was found vulnerable to catastrophic viral attacks: the compromise of a single machine during one election could affect results throughout the jurisdiction and potentially affect the results of future elections.

As of this writing, it has become clear that we cannot trust our elections to the electronic voting machines of today's leading vendors. Whether we will ever be able to trust them remains an open question. There is not yet a clear consensus on what standards a voting machine should reasonably be expected to meet. It is also by no means obvious that any set of feasible technical requirements would yield a voting machine worthy of our trust—it might simply be beyond the state of the art to create a sufficiently reliable and economical electronic voting machine. The point of this work is to make progress toward a better design, so as to bring us closer to understanding what is possible and to inform our standards and expectations for these machines.

Why does software correctness matter?

Switching from mechanical to electronic voting machines is a bigger step than it might seem at first. Today's electronic voting machines are not just electrically-powered devices performing the same function as their mechanical predecessors, the way electric light bulbs replaced oil-burning lanterns. Electronic voting machines contain general-purpose digital computers, which makes them fundamentally different and capable of much more than the special-purpose machines they replace. It would really be more accurate to call them "voting computers," as they are called in the Netherlands.

Just like any other general-purpose computer, a voting computer can be programmed to do anything—count votes, miscount votes, lie to voters, play games, or even attack other computers. To prove the point, a Dutch group called "Wij vertrouwen stemcomputers niet" ("We do not trust voting computers") reprogrammed the Nedap ES3B, their nation's leading voting computer, to play a passable game of chess [31].



Consequently, the types of attacks that are possible against voting computers are also fundamentally different than those possible against mechanical voting machines. Tampering with a lever machine can cause it to lose some votes or stop working entirely. Tampering with a computer can cause it to actively engage in sophisticated schemes to deceive voters and pollworkers, behave in different ways at different times or under different circumstances, and even subvert or conspire with other computers.

The behaviour of a general-purpose computer is determined entirely by its software. Assuring the correctness of software has been a major unsolved problem in computer science research for decades. Computer scientists have been able to prove some aspects of correctness for small programs, but all will readily acknowledge that nobody knows a general method for proving software programs to be correct. The software developed in industry tends to be larger and more complex

than can be analyzed by the best known techniques, while the programming languages and tools used in industry generally lag behind the state of the art in research.

Mistakes in software can remain latent for years, even when the code is publicly disclosed and inspected by motivated programmers. For example, OpenSSH is a popular program for secure login. Its developers have declared security to be their number one goal [17], and they have gained a reputation for security practices more rigorous than most. Nonetheless, security flaws were discovered in OpenSSH in 2003 that had been present since its first release in 1999, and had survived intensive software audits by the OpenSSH team.

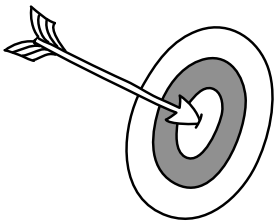
The problem is exacerbated by the possibility of *insider attacks*: what if someone involved in writing the voting software wants to bias the election? As far as anyone knows, the flaws in OpenSSH were inadvertent mistakes, so intentional flaws can probably be made even harder to find. (Chapter 8 offers some anecdotal evidence that detecting purposely hidden software flaws can be extremely difficult.) Reviewing the voting software is not just a matter of looking for code that seems intended to change votes or tallies. Any flaw that lets an attacker infiltrate the machine is a serious problem, since that flaw can then be exploited to reprogram the machine to do anything. So, a malicious programmer of voting machine software doesn't have to write suspicious-looking vote-altering code; he or she only needs to leave an innocent-looking security weakness. When a security weakness is found, there's no way to tell whether it is an intentional backdoor or an inadvertent mistake—as long as someone knows the flaw, it can be exploited. If any flaw can be an attack, we need voting software to be essentially flawless.

All of this explains why this dissertation focuses on software correctness. There are people who have many years of experience managing election personnel and running paper-based elections. There are people who know how to build reliable machines and reliable computer hardware. But the part that no one fully understands yet is how to get the software right.

2 Correctness

What constitutes a democratic election?

The democratic ideal of a legitimate election requires that the results reflect an unbiased poll of the voters—*accurate* according to what each voter intended, and *fair* in that each eligible voter has equal and unhindered opportunity to influence the outcome. These two basic goals can be broken down according to the mechanics of how elections are run.



Accuracy. By "accurate," I mean that the data about voter preferences is accurately gathered and combined to produce the final result. To make this happen, each ballot has to be processed correctly at the three stages of voting:

- **Correct ballot**: Each voter should be presented a ballot with complete and accurate information on the contests for which they are eligible to vote.
- **Cast as intended**: Each voter's recorded vote should match what the voter intended to cast.
- **Counted as cast**: The calculation that decides the outcome should accurately incorporate every recorded vote and no extraneous votes.

Fairness. By "fair," I mean that eligible voters (and only eligible voters) are free to vote as they please, without bias. We can look at this from two angles: how the sample of voters is drawn from the population, and how the opinions of the voters are measured.

- **Unbiased sampling**: Votes should come from a fair sample of the population of eligible voters.
- **Unbiased measurement**: Each vote should be a fair measurement of a voter's preference.

Each of these two aspects of fairness can be elaborated in further detail. In modern democracies, fair sampling is upheld through measures aimed at offering equal access to the polls, and also through the principle of "one person, one vote."

- **Unbiased sampling** is achieved by ensuring:
  - **Authorized voters**: Only voters that are eligible for a contest should be permitted to vote on it.
  - **One ballot per voter**: No voter may cast more than one ballot.
  - **Equal suffrage**: Every voter eligible for a contest should have an equitable opportunity to vote on it.

An unbiased measurement depends on eliminating influence from external pressures as well as influence from the presentation of the ballot itself.

- **Unbiased measurement** is achieved by ensuring:
  - **Secret ballot**: No voter's choices should be exposed by the voting system or demonstrable by the voter to others, lest votes be influenced by social pressure, bribery, threats, or other means.
  - **Equal choice**: Every option in a contest should have an equitable opportunity to receive votes.

Democracy also demands a further virtue: since power is derived from the consent of the governed, the election process itself must be accountable to the people. The manner in which all of the above goals are achieved should be **verifiable**, so that members of the public can assure for themselves that the election is accurate and fair. The verifiability of the election is not listed among the above goals because it is a "meta-goal," like a layer on top of all the other goals.

A widely preferred avenue for achieving verifiability is through transparency—exposing the election process to public scrutiny. However, verification can also take place through the investment of trust in independent experts or inspectors (or suitably balanced committees thereof), or through cryptographic means, in which a calculation provides mathematical evidence of the property to be verified.

What does it mean for a voting system to be correct?

In order to be confident that an election is democratic, we would want to have assurance of all of the goals just mentioned. But these goals are for the election as a whole, including all the people, processes, and technology involved. When we talk about a particular piece of equipment, such as a voting machine, we have to choose a specific set of subgoals that it is responsible for. For example, a voting machine cannot, by itself, guarantee that each voter only votes once. However, if the machine requires something like an access card in order to cast each ballot, this feature *in combination* with a suitably controlled process for handing out access cards, carried out by competent, trustworthy pollworkers, can effectively limit each voter to casting just one ballot.

Every goal is achieved through some combination of human processes and technology. This dissertation is primarily concerned with the technological part of an election—the equipment and software involved in collecting and counting votes, which I am calling the "voting system" for short. To say that the voting system works correctly means that it fulfills the responsibilities that have been assigned to it. Only after we've decided on this assignment of responsibilities is it meaningful to say whether it is correct. As the access card example illustrates, it is usually necessary to subdivide goals in some detail in order to separate out subgoals that technology can address.

How does correctness relate to safety?

Engineers have been designing safety-critical systems for many years, so it's instructive to examine the research and practice in methodologies for developing these systems.

Analysis. One of the most common analysis techniques for safety-critical systems is *fault tree analysis* [83]. Fault tree analysis is a way of identifying all the ways that a particular failure can occur. To perform fault tree analysis, one begins with a root node that represents the undesired event (the fault); then one identifies all the events or situations that could cause that undesired event, and each one becomes a child of the root node. Each node can be further refined by adding children that identify possible causes. For example, a few nodes in a fault tree for a fire extinguisher might look like this:

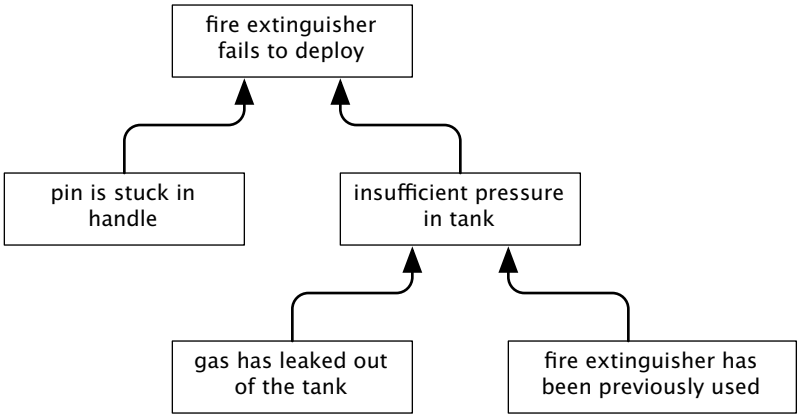


Figure 2.1. A small portion of a fault tree for a fire extinguisher.

Fault trees are known in the computer security world as *threat trees* [3] or *attack trees* [70]. An attack tree lays out all the possible ways that an attacker might come to violate a specific security restriction. In an attack tree, the top node is the attacker's ultimate goal. The children of a node specify various ways that an attacker can achieve the goal. For example, if the ultimate goal is to break open a safe, an attacker could do

so by obtaining the combination or by drilling open the safe. Part of the attack tree might look like this:

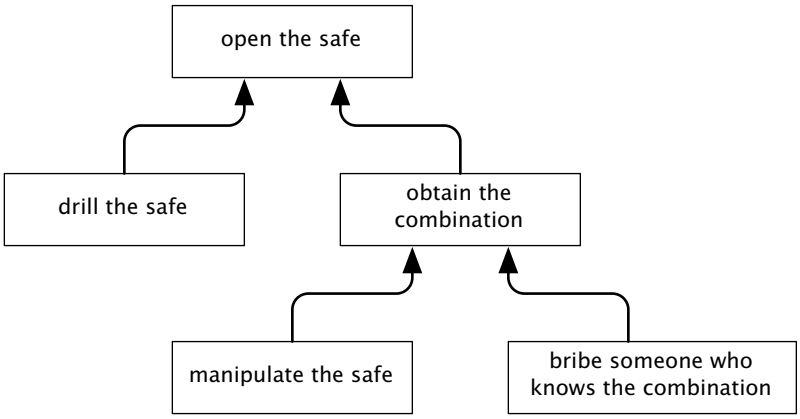


Figure 2.2. A small portion of an attack tree for an attacker who wants to break into a safe.

In the above examples, any one of the children of a node is sufficient to lead to the parent; the relationship among siblings is a disjunction (OR). Fault trees and attack trees can also specify conjunctions (AND) and other logical relationships. The nodes can be labelled with numbers to indicate the probability of an event or the cost of a step in an attack.

Design. Fault trees and attack trees are used to analyze existing systems to identify their weaknesses. But when one is designing a system, the goal is to establish the system's worthiness.

In the safety-critical literature, a written justification of a system's safety is called a *safety case* [87]. Safety cases are required by many safety standards. A safety case is often a very large document, as it incorporates all the arguments and supporting evidence for the safety of each element of the system. The development of the safety case can take up a large fraction of the effort in designing a safety-critical system. Hence, significant research efforts have been directed toward ways of organizing and maintaining safety cases.

Like fault trees, safety cases are also typically structured in a top-down approach based on successive refinement. The technique that is probably the most prominent in the research

literature is the Goal Structuring Notation [41], which elaborates on a basic tree-like organization of goals by allowing nodes of several different types: goals, strategies, justifications, assumptions, and so on. Here is an example of a section of a safety case for a microwave oven diagrammed in Goal Structuring Notation:

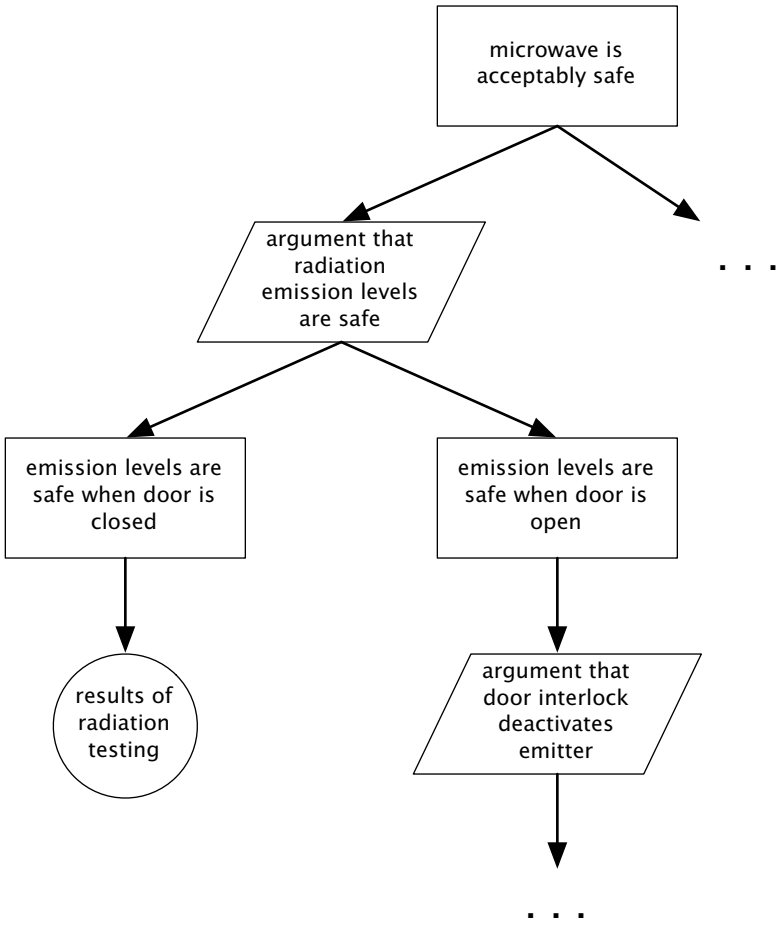


Figure 2.3. Part of a safety case for a microwave oven in Goal Structuring Notation.

Voting systems. A safety case would be appropriate for justifying why we should place our confidence in a voting system. Ideally, certification of any voting system for deployment would require the manufacturer to provide a convincing and clearly structured safety case.

The hierarchy of goals for a democratic election form the

starting point for such a safety case. The process of dividing goals into subgoals produces a tree of assurance goals for a system, which I'll call an *assurance tree*. (An assurance tree could be considered a simplified instance of Goal Structuring Notation in which all the nodes are correctness goals.) When an assurance tree is fully elaborated, the leaves of the tree are individual responsibilities that can be assigned to specific people and specific devices.

The process of refining the general goals into specific subgoals is a type of design activity. Different solutions will subdivide the main goals differently and assign responsibilities for the subgoals differently. For example, access cards are one possible way to keep voters from voting multiple times, but of course they are not the only way. It is a design choice to implement "one ballot per voter" in terms of the two parts: "pollworkers give one access card to each eligible voter" and "the voting machine allows each access card to be used just once to cast a ballot." Making these design choices and refining the goals at every level eventually leads to a set of specific technical requirements for the voting system.

In an assurance tree, the children of each node indicate what requirements have to be upheld in order for the parent goal to be upheld. The final result of refining the tree is an assignment of specific responsibilities to various parts of the system—for example, a set of tasks to be carried out by humans and a set of tasks to be carried out by computers—such that all the assurance goals are upheld. The tree captures the design of the system as well as the security assumptions that the designer made.

What is the tree of assurance goals for an election?

The requirements that were presented earlier can be refined one step further without specifying a particular voting system design. First I'll explain each subgoal, then present the whole tree, which can form a basis for the safety case of any election.

Accuracy: correct ballot. In order for a voter to receive a correct ballot, the correct ballot has to exist for that voter and it must contain the correct instructions and choices for the election. The voter then has to be given the right kind of ballot, and the voter has to receive it without alteration.

Accuracy: cast as intended. The voter's vote is properly recorded if the ballot indicates *what* the voter wanted and is cast *when* the voter is ready. Choices should be selected if and only if the voter makes them, and the voter should be free to mark the ballot in any manner that is valid. (When paper is used, the voter can also cast an invalid ballot; then the ballot is not counted. When electronic machines are used, the machine usually prohibits the voter from marking the ballot in an invalid manner.) To further ensure that the cast ballot matched the voter intent, the voter should get accurate feedback about what is currently selected, and should be able to make changes or corrections before casting the ballot.

Accuracy: counted as cast. For the count to be correct, there must be no extra or missing votes, and the votes that are counted must be exactly as voters indicated them on their cast ballots.

Fairness: authorized voters. I use the term *voting session* for the interval that begins with a voter entering a protected area of the polling place such as a voting booth, and ends when the voter walks away, either having cast or failed to cast a ballot. In

a typical election, voter authorization consists of controlling access to voting sessions and ensuring that there is no other way to cast a ballot except in a voting session.

Fairness: one ballot per voter. Limiting each voter to one cast ballot is also achieved by controlling access to voting sessions. In practice, each voter is authorized for one voting session at a time. If a voter wants to try again, a pollworker either destroys the ballot or determines that the voter did not already cast a ballot, and then authorizes another voting session.

Fairness: equal suffrage. There are three steps to casting a ballot. First the voter has to get to a polling station. Then, at the polling station, the voter has to be allowed to begin a voting session. Then, in the voting session, the voter has to successfully cast the ballot. Equal suffrage demands that voters have reasonable access and be free of discrimination at all of these stages.

Another way that a voter can be disenfranchised is to make an error. It is infeasible to demand that there be no errors at all, but fairness requires that errors not be biased against any particular group of voters. The controversy over the 2006 race for Florida's Congressional District 13 highlighted the significance of biased error. Different voters saw different ballot layouts, and post-election analysis [29] has suggested that the particular layout used in Sarasota County caused a large fraction of voters to skip the congressional race by mistake.

Fairness: secret ballot. The election system should not itself violate the voter's privacy. But it's a tougher task to prevent coercion. Voters' susceptibility to influence may not be based in reality: as long as voters *believe* they will profit or suffer by voting a certain way, the belief is sufficient to influence their votes. For example, an attacker could claim to have insider access that allows him to identify which voters voted for a particular candidate and punish them. Whether or not the attacker has such insider access, or whether discovering voters'

identities is even possible, the fear of punishment could be enough to sway votes. Different kinds of voting systems will lend differing degrees of plausibility to such claims—for example, some voters might be easily persuaded that someone could violate their privacy via computerized vote records, but they might find it harder to see how such a violation would be possible with hand-counted paper ballots.

The formal definitions of coercion-resistance in the research literature [18, 38, 60] require that voters be unable to *prove* to a vote-buyer that they voted a certain way. But the issue is more nuanced than that. A vote-buyer doesn't need solid proof, just evidence sufficiently plausible that offering a reward for it will influence the vote.

For example, consider an election system in which voters receive receipts indicating how they voted, but could also *forge* such receipts. One might think that such an election system is coercion-resistant, since it isn't worthwhile for a vote-buyer to buy something that can be forged. But resistance to coercion also depends on the cost of producing a forgery: if forgeries require enough effort that a significant number of voters will vote as directed by the vote-buyer instead of carrying out the forgery, the vote-buyer will succeed at influencing the election. Therefore, the secret ballot goal includes the requirement that voters not be given any *plausible evidence* (not just hard proof) of their votes that could be sold to an external party.

Fairness: equal choice. Since the goal is to avoid bias among the options within a contest, it would not do for some of the options to be shown one way to some voters and a different way (say, in red, or in larger print) to others.

It would be ideal to avoid all bias among options presented on the same ballot, but this is not possible: some option has to be presented first, and there is a well-documented bias toward the first item [46]. The next best thing is to change the order of presentation from ballot to ballot such that there is a uniform *distribution* of bias towards all the options, when the ballots are considered in aggregate.

There is a more subtle kind of bias that also should be avoided: a bias relative to the voter's preferred choice. Imagine, for example, a contest with three options A, B, and C. Suppose the ballot design causes half the voters who intend to mark A to mistakenly mark B, half of those who want B to mark C, and half of those who want C to mark A. Such a ballot is not biased toward any particular option, but it is still clearly unfair: B could win an election in which most voters intended to vote for A. So there is also a requirement for a uniform distribution of errors with respect to the voter's intended choice.

* * *

Gathering all the requirements just mentioned gives us the following high-level assurance tree for elections.

Accuracy

- **Correct ballot**
  - G1. For every voter, there exists a ballot style containing the complete set of contests in which that voter is eligible to vote.
  - G2. On every ballot, all the information is complete and accurate, including instructions, contests, and options.
  - G3. In every voting session, the correct choice of ballot style is presented to the voter.
  - G4. Every ballot is presented to the voter as the ballot designer intended.
- **Cast as intended**
  - G5. At the start of every voting session, no choices are selected.
  - G6. The voter's selections change only in accordance with the voter's intentions.
  - G7. The voter receives accurate feedback about which choices are selected.
  - G8. The voter can achieve any combination of selections that is allowable to cast, and no others.

G9. The voter has adequate opportunity to review the ballot and make changes before casting it.

G10. The ballot is cast when and only when the voter intends to cast it.

- **Counted as cast**

G11. Every selection recorded on a ballot cast by a voter is counted.

G12. No extra ballots or selections are added to the count.

G13. The selections on the ballots are not altered between the time they are cast and the time they are counted.

G14. The tally is a correct count of the voters' selections.

Fairness

- **Unbiased sampling**
  - **Authorized voters**

G15. Only authorized voters can begin voting sessions.

G16. Only in voting sessions can ballots be cast.

  - **One ballot per voter**

G17. No voting session allows more than one ballot to be cast.

G18. Each voter is allowed at most one voting session in which a ballot was cast.

  - **Equal suffrage**

G19. Every voter has reasonable, non-discriminatory access to a polling station they can use.

G20. Every voter can begin a voting session within a reasonable, non-discriminatory waiting time.

G21. Every voting session provides a reasonable, non-discriminatory opportunity to cast a ballot.

G22. For every voter that is eligible to vote in a particular contest, there is a uniform likelihood of voter error on that contest.

- **Unbiased measurement**
  - **Secret ballot**

G23. The processing of voter choices does not expose how any particular voter voted.

G24. Voters are not provided any way to give plausible evidence of how they voted to an external party.

- **Equal choice**
  - G25. Within each contest, all the options are presented in the same manner on each ballot and across all ballots.
  - G26. For each contest, the voters are presented with ballots that, in aggregate, yield a uniform distribution of bias in favour of each option.
  - G27. For each contest, the voters are presented with ballots that, in aggregate, yield a uniform frequency of voting errors across the voters that intend to vote for each option.
  - G28. In each contest, for each option, voters intending to vote for that option are presented with ballots that, in aggregate, yield a uniform distribution of voting errors in favour of every other option.

What does it mean for a voting system to be secure?

A voting system is secure if it can be relied upon to produce the correct results in the face of determined attempts to corrupt the outcome. Thus, security and correctness are closely related: security is just correctness in an adversarial context. The intentional violation of any subgoal in the assurance tree would constitute a security breach.

Since this dissertation is focused on the software security questions surrounding electronic voting machines, let's separate out the goals that rely on software from those that don't.

Of the goals in the assurance tree, these are normally addressed by humans in the preparation and conduct of the election:

- G1. For every voter, there exists a ballot style containing the complete set of contests in which that voter is eligible to vote.
- G2. On every ballot, all the information is complete and accurate, including instructions, contests, and options.
- G18. Each voter is allowed at most one voting session in which a ballot was cast.
- G19. Every voter has reasonable, non-discriminatory access to a polling station they can use.

The following goals are addressed through good ballot design. They could be violated by voting machine software that displays the ballot incorrectly or lacks the ability to display ballots in a fair manner. However, as long as the voting machine presents the ballot as the ballot designers intended (which is goal G4), we can consider these goals the responsibility of ballot designers:

- G22. For every voter that is eligible to vote in a particular contest, there is a uniform likelihood of voter error on that contest.
- G25. Within each contest, all the options are presented in the same manner on each ballot and across all ballots.

G26. For each contest, the voters are presented with ballots that, in aggregate, yield a uniform distribution of bias in favour of each option.

G27. For each contest, the voters are presented with ballots that, in aggregate, yield a uniform frequency of voting errors across the voters that intend to vote for each option.

G28. In each contest, for each option, voters intending to vote for that option are presented with ballots that, in aggregate, yield a uniform distribution of voting errors in favour of every other option.

The following goals could be addressed almost entirely by election-day procedures, or through a combination of such procedures and proper software behaviour, depending on how the voting system is designed:

G15. Only authorized voters can begin voting sessions.

G16. Only in voting sessions can ballots be cast.

The proposed designs in this dissertation assume that the above two goals are upheld by human procedures. For G15, election workers ensure that only authorized voters are permitted physical access to voting machines. And for G16, election workers should provide no other way to cast ballots outside of the officially approved procedures.

The remaining goals are those that necessarily depend on the correctness of the voting machine software implementation:

G3. In every voting session, the correct choice of ballot style is presented to the voter.

G4. Every ballot is presented to the voter as the ballot designer intended.

G5. At the start of every voting session, no choices are selected.

G6. The voter's selections change only in accordance with the voter's intentions.

G7. The voter receives accurate feedback about which choices are selected.

G8. The voter can achieve any combination of selections that is allowable to cast, and no others.
G9. The voter has adequate opportunity to review the ballot and make changes before casting it.
G10. The ballot is cast when and only when the voter intends to cast it.
G11. Every selection recorded on a ballot cast by a voter is counted.
G12. No extra ballots or selections are added to the count.
G13. The selections on the ballots are not altered between the time they are cast and the time they are counted.
G14. The tally is a correct count of the voters' selections.
G17. No voting session allows more than one ballot to be cast.
G20. Every voter can begin a voting session within a reasonable, non-discriminatory waiting time.
G21. Every voting session provides a reasonable, non-discriminatory opportunity to cast a ballot.
G23. The processing of voter choices does not expose how any particular voter voted.
G24. Voters are not provided any way to give plausible evidence of how they voted to an external party.

G3 and G20 depend on election-day procedures as well as the voting machine software. For G3, typically a pollworker is responsible for selecting the correct ballot style for each voter, and the voting machine must correctly use the ballot style indicated by the pollworker. For G20, the polling station needs to serve voters efficiently and fairly, but also the voting machines should be available and ready to serve voters and should not freeze up or crash. G23 and G24 depend on the overall design of the voting system, including the human procedures, as well as the correct functioning of the voting machine software.

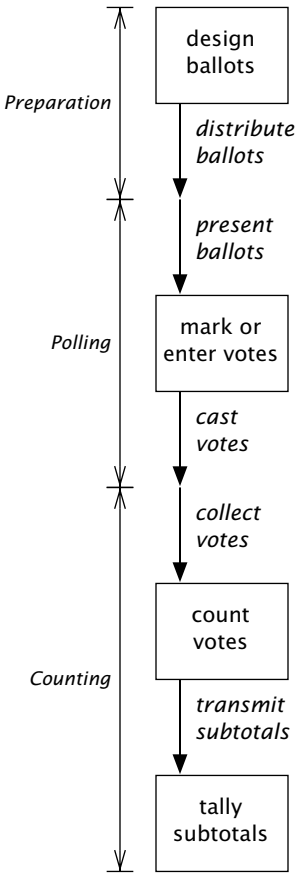
Security issues with voting machine software usually have to do with upholding and enforcing the 17 goals in this last list. These 17 goals are the focus of my efforts to achieve and verify software correctness.

3 Verification

How do we gain confidence in election results?

An election consists of many steps, each of which processes information such as ballot and candidate data, voter information, and records of cast votes. At the most basic level, each step takes some input and produces some output. Confidence in the ultimate result—the output of the last step in the chain—depends on confidence that each step was correctly performed. The choice of the type of voting system determines which steps are carried out by people and which by computers.

Earlier we described the election process in terms of three stages: preparation, polling, and counting. With respect to establishing confidence in a voting system, these stages can be broken down further into the nine steps shown at the left, which include transmission as well as processing of information.



The *preparation* stage consists of events prior to the opening of polls, which includes not only designing the ballots but also distributing them to polling places. This production and distribution takes place for both paper ballots and electronic ballot definition files.

The *polling* stage involves presenting the ballots to voters, who make selections and cast the ballots. For sighted voters reading paper ballots, presentation of the ballot is a trivial step, but for electronic voting computers the fidelity of the presentation is a real issue.

In many elections, *counting* occurs in two parts: votes are first counted at polling places, then the counts are centrally tallied to yield the final results. This stage includes the transmission of votes to the person or machine that counts them. The distinction between local and central counting is important because the local counting process often takes place in public, whereas the aggregation of results and central tallying does not.

For a step that transforms information from one form to another, confidence comes from ensuring that it produced the correct output for the input it was given. For a step that

transports information from one place to another, confidence comes from ensuring that the integrity of the information was preserved.

Because of the way I've defined the three accuracy goals (*correct ballot*, *cast as intended*, and *counted as cast*), they differ slightly from the three chronological stages: getting the correct ballot to the voter includes the presentation step at the polls. The following figure shows which steps correspond to the three accuracy goals. Under each step is the name of a subgoal for that step.

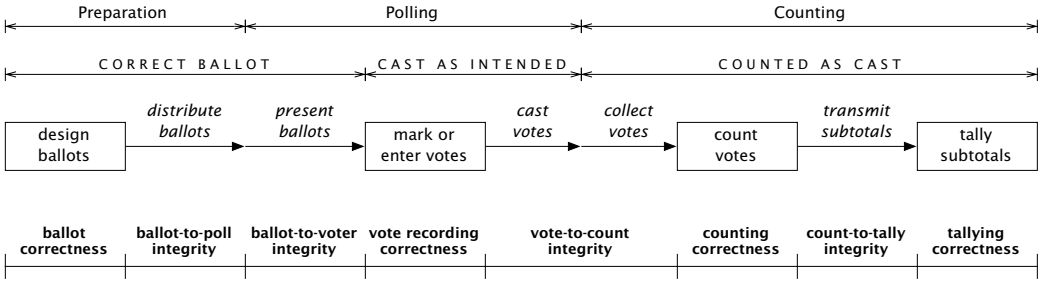


Figure 3.1. The nine steps in the election process and their corresponding integrity and correctness goals.

How can we verify the computerized parts of an election?

Suppose that a particular information processing step in an election is carried out by a computer. As I mentioned in Chapter 1, the computer's behaviour is completely controlled by its software. Let's say the software program responsible for this step takes some input *x* and produces some output *y*. For example, if this is the vote-tallying step, *x* could be a collection of electronic vote records and *y* could be the election totals.

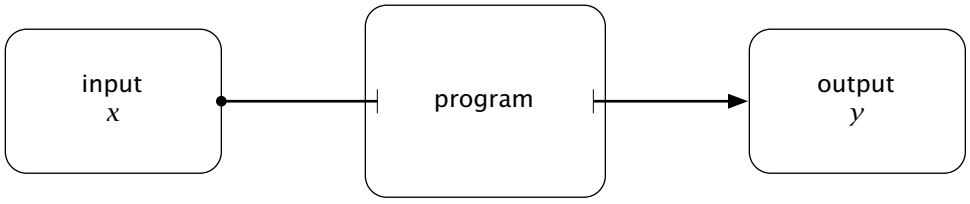


Figure 3.2. For some particular processing step in an election, a software program takes the input *x* and produces the output *y*.

If you want to check that the program produced the correct result, you have two main choices:

1. *Software verification.* You can examine the program itself and confirm that it works the way you expected. Depending on the assumptions you make, this may include manual inspection of the source code, automated analysis, or formal mathematical proofs. Once you have confirmed that the program does exactly what it's supposed to do in every possible circumstance, you can be confident that this particular output, *y*, is correct.
2. *Result verification.* You can take the input *x* and figure out what the corresponding output should be. If the actual output *y* matches the expected output, then you know it's correct. To do this, you need records of both *x* and *y*, as well as some way to independently repeat the operation—perhaps you have another program that you trust, or perhaps you can work out the expected output by hand.

There is also a variant of result verification:

2a. *Indirect result verification.* Some schemes allow you to establish confidence without repeating the entire operation. For example, given information *derived* from *x* and *y*, you might have a way to mathematically check their consistency. Or, you might be allowed to choose *parts* of *x* and *y* to check, enabling you to establish a high *probability* of a correct result.

Software verification has the advantage that it only needs to be done once on a given program to establish confidence in all the output it will ever produce. Result verification has to be repeated each time the program produces new output. However, there are three major factors weighing in favour of result verification.

Programs change. The apparent advantage of doing software verification only once becomes less compelling when you consider that software changes all the time. Features are added; bugs are discovered and fixed; demands change. In particular, election software is subject to election law, which differs from state to state in the United States. Whenever legislation gets passed, election software may have to be updated to satisfy new requirements. Any change would invalidate previous reviews or proofs of correctness and require the software to be verified over again.

Software verification requires disclosure. Disclosure of software code often faces legal, financial, or political barriers. Voting machine companies have resisted public disclosure of their source code on the grounds that it could help a motivated attacker, and they claim that copyright and trade secret protection are necessary to support a sustainable, profitable business. [34] Disclosing code would certainly increase the transparency of an election and improve the accountability of the testing process. But having ways to check the correctness of an election without *depending* on disclosure of all the code would allow the election to sidestep this disclosure dispute. The

democratic process is healthier if private interests have fewer opportunities and fewer plausible incentives to prevent the public from verifying an election.

Software verification is much harder. As a later section of this chapter will explain (page 41), the behaviour of software can be extremely difficult to analyze. Software review by human experts is expensive, time-consuming, and prone to error. The only way to be truly sure is to construct a mathematical proof, but it is well beyond the state of the art to do this for programs the size of typical computer applications. When such proofs are constructed, they often aim to prove things about a simplified model of the program rather than the program itself. Unfortunately, a mathematical proof can only prove that a program satisfies a formal specification of what it's supposed to do. The proof only establishes that the program is correct if the specification accurately expresses what it means to be correct—and such specifications are themselves complex and tricky to write.

What kind of election data can be published?

There is an inherent tension between voter privacy and the desire for verifiable elections. As argued earlier in this chapter, verifying results is preferable to verifying software. But public verification of results depends on publishing election data.

Suppose there is some data made available to the public to enable verification. This might include partial or complete information about ballots, votes, and results, or something derived from such data. Each published piece of data (let's call it a *record*) might be *identifiable* as corresponding to a particular voter, or it might not. And each record might contain sufficient information to *reveal votes*, or it might not. These two features are independent: for example, a published record could indicate a vote for a particular candidate, yet not be associated with any particular voter.

For voters to be able to check that their own ballot was correctly received (i.e., cast as intended), they need to be able to look up their own ballots. To do this, they need some kind of public record of their ballot that is *identifiable*.

For voters to be able to confirm the tally by directly performing their own recount, they have to be able to see the votes. To do this, they need public records that *reveal votes*.

Published records that are identifiable *and* reveal votes would enable the public to verify everything, at the expense of voter privacy. Imagine an election in which every ballot is published online and uniquely associated with the voter who cast it. Any voter could look up their ballot online to confirm that it is correct as published, and anyone could count the published ballots to confirm the tally. In such a system, software correctness would be irrelevant—software could be used at any stage of the process and there would be no need to verify it, because the entire election can be checked by result verification. But in such an election, voters could also easily sell their votes—for example, they could tell a vote-buyer where to find their ballots online.

* * *

In summary:

- Public confirmation that ballots are cast as intended requires public records that are *identifiable*.
- Public confirmation of the tally by direct recount requires public records that are *vote-revealing*.
- If any public records are identifiable *and* vote-revealing, they enable bribery and coercion.

This suggests two possible kinds of public records:

1. Anonymous records that do reveal votes.
2. Identifiable records that don't reveal votes.

Several proposals for voting systems, including those proposed in this dissertation, publish records of the first kind. These records enable direct result verification of the tally. Later in this chapter, I'll discuss end-to-end cryptographic voting systems, in which both kinds of records are published, and an additional verification step confirms the correspondence between the two.

What makes software hard to verify?

Most software is hard to verify because it is complex.

Here are some of the main reasons why complexity in software is more difficult to manage than complexity in a physical machine.

Number of components. The number of parts in a physical machine is limited by the costs of manufacturing, but there is no such limit on software. A software program costs the same to distribute—virtually nothing—whether it contains ten components or a million components. It is easier to add complexity to a software program than to a physical device, and removing code often has a higher risk of breaking the program than adding new code. Requirements change and customers ask for more features; in response, software tends to grow boundlessly during the course of development, unless there are determined and persistent efforts to keep it small.

Software programs also often incorporate large ready-made packages of components written by others, to save the effort of writing code from scratch. Even if only a small part of a package's functionality is used, it is easier to include the entire package than to separate the parts that are used from those that are not. These pressures lead to software applications with millions of lines of code and thousands of interacting components.

Complex interconnections. There are likely to be more connections between the parts of a software program than those of a physical machine. Whereas a machine part can only interact with other parts near it, there is no limit on the number of other parts that a software component can depend on. For example, it is common for a single component to be relied upon by thousands of other components.

These connections are also harder to see in software. The way that a machine part affects other parts is usually clear from

direct physical inspection. But finding all the other software components that depend upon a given software component can be a difficult task.

Far-reaching effects. Because software components can be so deeply interconnected, a small change in one part can affect another part that is far away, affect parts written by different people, or have wide-ranging effects on the behaviour of the whole program. The software engineering practices of modularity (dividing up a program into distinct modules) and encapsulation (protecting each module from outside interference) aim to limit these kinds of effects, but software programs nonetheless tend to be more sensitive to change than physical machines.

Nonlinearity. The power of general-purpose computers derives from their ability to make decisions. With software, a tiny change in input can yield a completely different outcome; for example, a program can decide to behave one way when the result of a calculation turns out to be zero and another way when it is nonzero. This means that similar situations cannot be assumed to yield similar behaviour. This *nonlinear* nature makes it hard to predict how software will behave and hard to test software thoroughly. Mechanical devices can be nonlinear too, but software tends to be pervasively nonlinear.

* * *

One of the most serious threats that is currently poorly addressed in voting systems is the insider threat from software developers. Intentionally placed bugs or backdoors are hard to detect even when software is carefully audited [5]. The persistent failure of the federal testing process to detect major security flaws [21, 37] and the continuing revelations of security vulnerabilities in certified voting systems [33, 43, 64, 84, 88] suggest that voting software has not been audited anywhere near enough to defend against this threat.

The complexity of software is what makes it difficult to be *sure*: sure that the software will behave as expected, that it will produce the correct results, and that it will resist determined attempts to subvert the outcome of an election. Software complexity is the ultimate enemy of reliable computer-based elections.

There are two ways to fight this enemy: design the system so less of the software needs to be verified, and simplify the software that needs to be verified. Both can be applied together.

In what ways are today's voting systems verifiable?

Different voting systems offer different ways for voters to gain confidence that the election results are correct. We can compare systems by looking at what mechanism for assurance is provided, if any, at each step of the process.

The two kinds of voting technology most commonly used in the United States are *optical scan* systems and *direct recording electronic* (DRE) systems.

Optical scan voting. When an election is conducted by optical scan, paper ballots are prepared and printed before polls open. Voters mark the ballots by hand and deposit them into a ballot box. There are two variants of optical scan voting: the scanning can take place at individual precincts or at a central election office.

Although software is usually involved in preparing the ballots, voters and candidates can verify for themselves the sample ballots published before polling. Voters can also bring sample ballots to the polling place and compare them with the blank ballots they receive. This is an example of avoiding software verification, which is possible because the results of the preparation stage are public.

We know the ballot is presented exactly as prepared, because the voter directly reads the printed paper. There is no recording device to misrecord the voter's marks; the voter is responsible for clearly marking the paper to be counted. The election relies on the physical durability of paper for the integrity of printed ballots and recorded votes.

A precinct-based optical scanner.

When scanning takes place at individual precincts, the ballots pass through a scanning machine on their way into the ballot box. After polls close, each machine prints out its counts on a paper tape. If the paper tapes are posted immediately for public viewing, then no one has to trust the software that does the tallying. The final election report will contain both the

counts in each precinct and the overall totals. Anyone can confirm that the locally posted results are correctly included in the election report, and anyone can confirm that the overall totals were calculated properly.

When scanning is performed centrally, voters can't perform the same check on the tally step. They have to trust election personnel to safely transport the ballots from the polls to the central office and to enter the results from the central scanner into the software that tallies them (known as the election management system, or EMS).

Figure 3.3 summarizes the mechanisms by which any individual voter can ensure the validity of each step in this process. (I'll call this an *assurance chart*.)

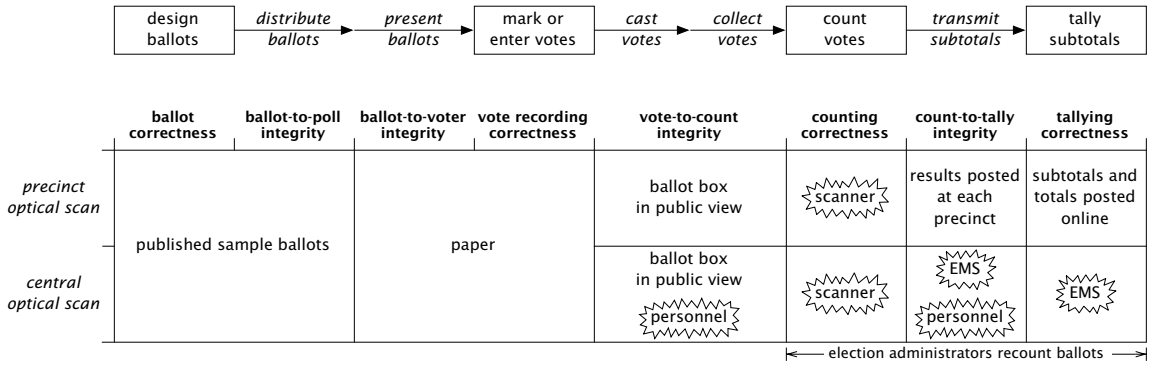


Figure 3.3. Assurance chart for elections with hand-marked, optically scanned ballots.

The starbursts mark mechanisms that voters have to accept on faith—they have to trust software they can't see or people they don't know. For precinct-based scanning, voters have to trust the software that controls the optical scanner. For central scanning, the voters also have to trust the personnel who collect the ballots and convey counts from the scanner to the EMS. They also have to trust the EMS itself, since they have no way to independently check that the totals were added up correctly.

Paper ballots provide a useful backup record, as they can be recounted by hand or by machine. The same stack of ballots can even be counted multiple times, and the counts from different people or different machines can be compared to improve

confidence. In Figure 3.3, recounts are shown as a secondary assurance mechanism, below the three boxes on the right. They are shown as secondary because ordinary voters cannot conduct or order recounts; only election administrators can do so.

DRE voting. Figure 3.4 shows what voters have to trust for each step of an election process with a DRE voting system. There are two possibilities here as well: the results from DRE machines might be reported at each precinct, or they might be reported only by the central election office.

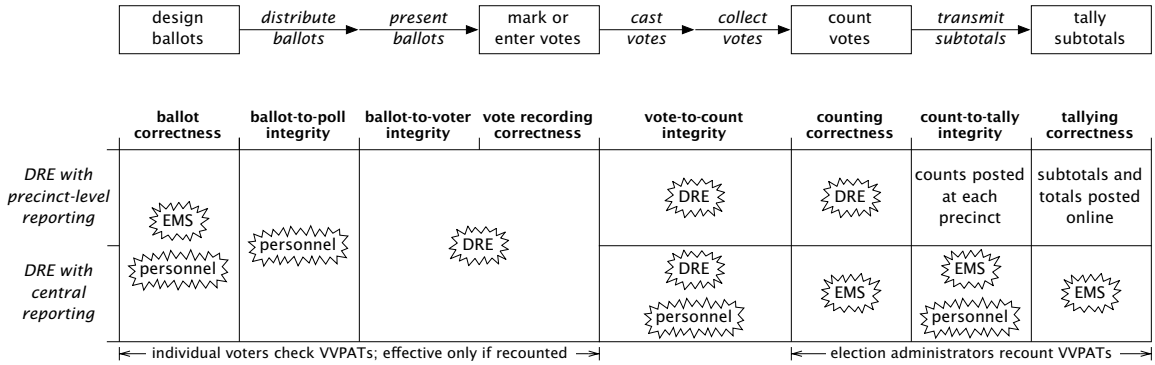


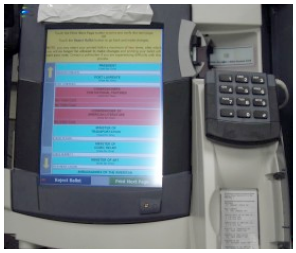
Figure 3.4. Assurance chart for elections with direct recording electronic (DRE) voting.

When DRE machines are used, voters don't get to see a sample of the ballot definition in the machine, in the same way that a sample ballot is a direct preview of what will be used on election day. At best, voters might get images of the screens displayed by the DRE, printed on paper. But, in general, they don't get to test-drive a DRE with the ballot definition they will be using, and they can't check whether their machines have received the correct ballot definitions. Voters have to trust the EMS, which produces the ballot definition files, the personnel that operated the EMS, and the personnel that loaded the ballot definitions into the DRE machines.

A DRE voting machine.

The DRE machines are responsible for presenting the choices to the voter and recording the voter's selections. For these steps the voter is forced to trust that the DRE software is correct. For the counting stage, voters have to trust either the

software in the DRE that counts and reports results locally, or the software in the EMS that counts and tallies the results centrally, along with the personnel that convey the information to the EMS.



A DRE with a VVPAT printer (at lower right).

As a backup verification mechanism, some DRE machines print *voter-verified paper audit trails* (VVPATs). This is a paper tape that shows the voter's selections for viewing and confirmation by the voter. Printed VVPATs are retained by the machine so that they can later be recounted if a recount is deemed necessary. However, voter inspection of VVPATs is not as strong a backup as voter inspection of paper ballots; in the case of VVPATs, the thing being inspected is not what is normally counted. With DRE machines, the results are derived from the electronic records, not the VVPATs that voters see; the VVPATs are only relevant if election officials decide to conduct a recount.

There are also good reasons to believe that voters are unlikely to catch discrepancies on VVPATs. In a study by Everett [25], voters using a mock DRE were shown a review screen with selections different from what they had chosen, and 68% of voters failed to notice the changes. It seems likely that even more voters would miss discrepancies on the VVPAT, which is generally smaller than the screen and shown off to the side of the machine.

As Figure 3.4 makes obvious, DRE voting systems depend heavily on software. Because so little information is typically published about these programs and their inputs and outputs, trusting the outcome of such an election often requires trusting virtually every piece of software in the system—software for designing ballots, software that produces ballot definitions, voting machine software, software that tallies votes, and all the operating systems, compilers, editors, and other tools that were used to produce these programs.

It doesn't have to be this way. By publishing information about the software and the data processed by that software, it's possible to reduce what voters have to accept on faith in order to trust the validity of the election result.

What is the minimum software that needs to be verified?

The degree to which software verification is avoidable depends on a critical decision: how do voters indicate their votes—on paper or on a computer? Of all the steps in the process, this one is special because it must take place in private.

A big part of the present controversy over electronic voting machines is a conflict about the user interface presented to voters. Proponents of the machines point to the real benefits that computers could offer in improved usability and accessibility. For people with certain disabilities, voting computers may be the only way to vote privately and independently. Whether these advantages are enough to outweigh the loss of a tangible, directly marked ballot is a complicated question, and I argue for neither side of that issue here. But an important factor in deciding whether vote entry should occur on paper or on a computer is the feasibility of ensuring the integrity of votes in either case.

Each of the two cases has its own answer to "what is the minimum software that needs to be verified?"

Case 1: The paper option. If voters directly mark paper ballots, the answer is "nothing." To avoid all software verification, just publicly count the ballots by hand right after the polls close. Sample ballots, mailed out before polls open, let voters check that the real ballots are printed correctly. There is no software involved in marking and casting votes, only paper. And if the results of the hand count are posted immediately at the polling place, then no one has to trust the software that does the tallying.

So, in a voting system where paper ballots are hand-marked and hand-counted at the polls, any step that uses software can be publicly checked by direct result verification. As with any paper ballot system, the ballots are available to be recounted later if necessary.

Figure 3.5 summarizes the preceding analysis in an assurance chart.

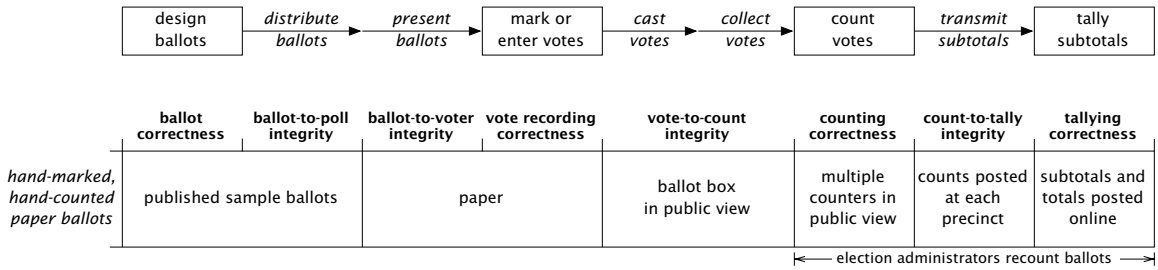


Figure 3.5. Assurance chart for an election with hand-marked, hand-counted ballots.

Case 2. Entering votes by computer. In this case, the answer is "just the vote-entry software." Here's why.

The "mark or enter votes" step, central to the voter experience, also turns out to be critical in terms of verification. This step cannot be publicly verified by result verification. Result verification requires a complete record of inputs and outputs. But one of the inputs to this step is the input from individual voters, which must be kept private due to the principle of the secret ballot. Moreover, if the ballot is presented to the voter by a computer, the voter's input is subject to influence by the computer.

Therefore, if choices are presented or selected on a computer, software verification is unavoidable. However, the secret ballot is the only privacy requirement that elections have to uphold. Recorded votes can be published as long as they cannot be associated with any particular voter. **The only part of the process that needs to be secret—and thus the only part for which software verification is really necessary—is from the private interaction with an individual voter up until the moment the voter's votes are recorded in anonymous form.** That interval is the critical interval during which private information gets turned into publishable information. All the inputs and outputs for other steps can be published, so everything else can be checked by result verification.

It follows that the way to minimize software verification is to make that critical interval as short and simple as possible: use software to present the ballot, accept selections from voters, and record the votes in anonymous form, then publish the anonymous votes immediately when polls close. The preparation that takes place before the election produces a ballot definition file for the voting machine. If this file is also published, no one needs to verify the ballot preparation software either. Figure 3.6 gives the assurance chart for this case.

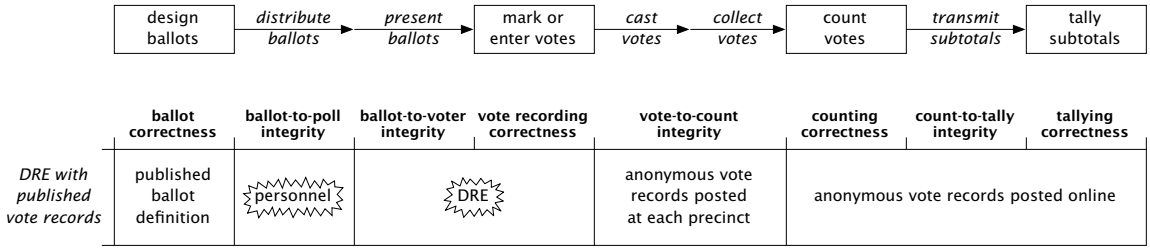


Figure 3.6. Assurance chart for a DRE-based election with published ballot definition and published, anonymous vote records.

In the ballot distribution step, voters have to assume that election personnel have properly distributed the ballot definitions and loaded them into the machines; they have no way to check this for themselves. And in the ballot presentation and vote recording steps, voters still have to trust the software in the DRE machine.

Practical example. Here's one way that an election with computerized voting but minimal software verification could be carried out in practice.

The software for the voting computer would be written to run on a free computing platform, and finalized and published far in advance of the election so that everyone has time to inspect it and test it. The ballot definition files for the election would be published on government websites, also far enough in advance that members of the public have time to examine them

before the polls open. Anyone would be able to download a ballot definition and run the voting computer software on their own computer to see exactly what will be shown to voters on election day. This provides a chance to detect omitted races, misspelled candidate names, layout errors, and other ballot errors. Thus, the published ballot definition file serves a similar purpose to the paper sample ballot typically mailed to voters before an election.

When a polling place stops accepting new votes at the end of the day, each machine should contain a vote file containing all of its anonymously recorded votes. At this point, every machine would print out a *cryptographic hash* of its vote file; observers can copy down (or photograph) the hashes. A cryptographic hash is a number derived from the contents of a file in such a way that it is easy to calculate the hash for a given file, but difficult to produce a different file that yields the same hash. Publishing the hash makes a public commitment to the contents of the file. (The reason for using a hash is that it is less cumbersome than printing out the entire vote file, but it serves the same purpose.)

The anonymous vote files from every machine would then be published online for all to see after the election. Anyone can calculate the hashes of these files and compare them to the hashes that were printed on election night, to verify that the files are authentic and unaltered. And anyone can count the votes in these files to confirm that the tallying is performed correctly.

The consequence is that neither the ballot layout software nor the vote tallying software would need to be verified. The published ballot definitions, voting computer software, and anonymous vote records would be sufficient to allow members of the public to independently check the accuracy of the election outcome.

What other alternatives for verification are possible?

Electronic ballot markers and printers. An *electronic ballot marker* (EBM) is a computer that marks a paper ballot [80, 81]. The voter inserts a paper ballot and makes selections on the computer, and the EBM prints marks onto the ballot in the appropriate positions. An *electronic ballot printer* (EBP) is a computer that prints out a marked paper ballot. No ballot is inserted; the voter makes selections on the computer, and the EBP prints out a fresh paper ballot that indicates the voter's choices. In both cases, the voter then deposits the paper ballot into a ballot box as usual.

EBMs and EBPs occupy a middle ground between optical scan systems and DRE systems. They provide the flexibility of a computerized user interface for voting, together with a durable paper record that can be recounted later. Like a DRE machine, an EBM or EBP relies on a ballot definition file to describe the choices to present to the voter, and the proper recording of the voter's choices depends on the software running in the EBM or EBP. But the voter now has the option of checking the printed ballot before casting it, instead of having to trust this software. And unlike the printed VVPAT produced by a DRE, this printed ballot is always counted, so the voter's check is more effective.

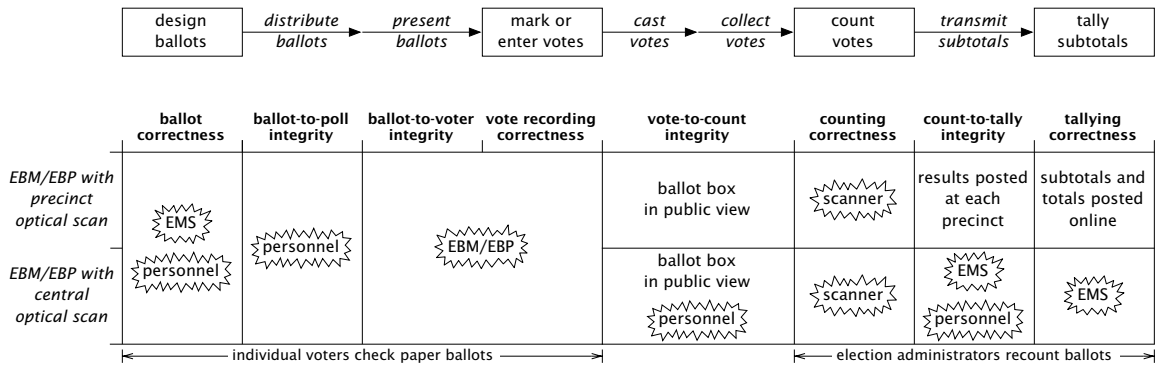


Figure 3.7. Assurance chart for an election with electronically marked or printed, optically scanned ballots.

The corresponding assurance chart, in Figure 3.7, has a left half similar to that of a DRE system, and a right half similar to that of an optical scan system.

End-to-end cryptographic voting. There are several proposed voting systems that provide *end-to-end cryptographic* methods for letting voters verify the election. "End-to-end" refers to the ability of any individual voter to check that his or her ballot survived from one end of the process straight through to the other—from casting to the final result—without special access from election officials.

Recall that earlier in this chapter, I described two possible kinds of publishable records—anonymous vote-revealing records, and identifiable but non-vote-revealing records. End-to-end cryptographic schemes publish records of the second kind as well as the first kind. Examples of these schemes are Punchscan [26], Scratch & Vote [1], Prêt-à-Voter [13], and VoteHere [54]. What they all have in common is that they publish *some* information about each voter's ballot: enough to let the voter partially check the recorded ballot, but not enough to reveal an actual vote so a voter can sell it. That is, *indirect result verification* is used to ensure the integrity of individual ballots. The partial records are set up in such a way that, with enough voters checking this partial information, the likelihood of an incorrectly posted ballot is nearly zero.

In addition to the partial ballots, actual vote records are separately posted—but these votes have been shuffled so they cannot be associated with particular voters. Anyone can count the posted votes to check the tally. The shuffling is performed using a system called a "mix net," in which multiple parties participate in the shuffling; no single party learns the total shuffling order, and thus voter privacy is protected.

In these end-to-end cryptographic schemes, the election authorities keep some secret information that enables them to process the ballots into verifiable totals, and the ballots contain serial numbers or cryptographic information as well. In all of these schemes, there is a pre-election audit procedure that lets

voters ensure that this information is consistent and properly formed. After the election, voters can also audit the shuffling procedure to confirm that the posted partial ballots correspond to the posted anonymous vote records, and thus to the tally.

The same mathematical techniques can be applied to votes cast in any fashion (by hand-marked paper, by machine-marked paper, or directly by machine). When hand-marked paper is used, the election can completely escape dependence on software. Figure 3.8 summarizes how assurance is provided in this category of systems.

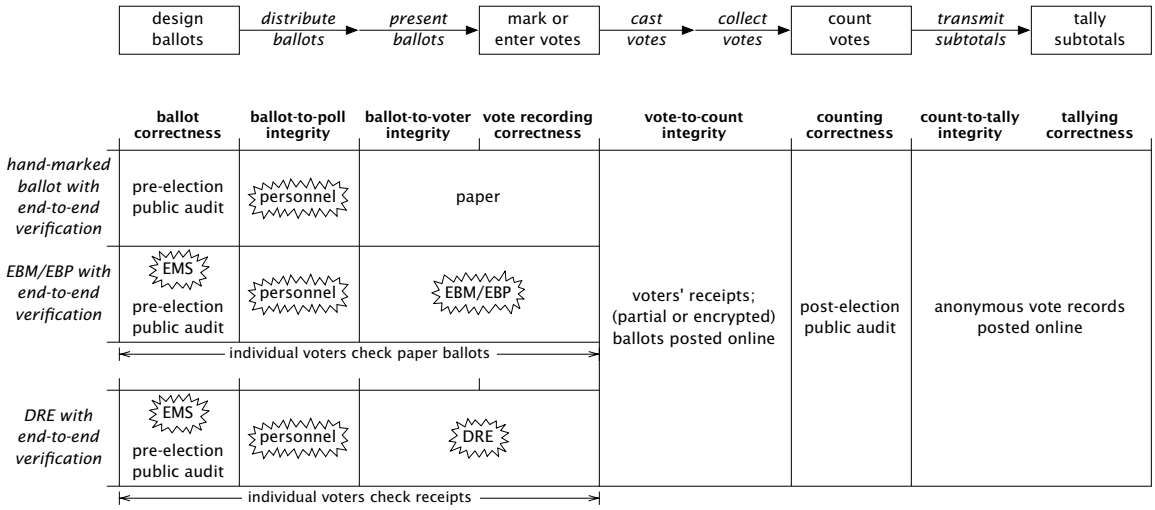


Figure 3.8. Assurance chart for elections with end-to-end cryptographic verification.

Non-cryptographic end-to-end schemes. Of special note are ThreeBallot, VAV, and Twin [67], which provide end-to-end verification without cryptography. These schemes publish all the cast ballots, which anyone can recount to verify the tally. In ThreeBallot and VAV, only *some* of the posted items are identifiable. Each voter's ballot is split into three parts; although all the parts are posted, the voter gets a receipt for only one part—and a single part isn't enough to reveal how they voted. In Twin, each voter gets a receipt for *someone else's* ballot. Thus, while the posted records can be matched with receipts, they can't be identified as belonging to any particular voter. The

assurance chart for all these schemes is similar to Figure 3.8, except there is no need for a post-election cryptographic audit because no encryption or shuffling has taken place.

Comparing voting systems. Figure 3.9 summarizes several types of voting systems on a single chart for comparison.

For conventional paper-based systems, shown at the top, any method of marking ballots (by hand, by EBM, or by EBP) can be combined with any method of counting ballots (by hand count, by precinct optical scan, or by central optical scan). Next come the conventional electronic systems, based on DREs; then the end-to-end cryptographic systems. Finally, at the bottom is the DRE with its ballot definition and results published, as well as a variant of the same scheme using an EBM or EBP instead.

The systems least dependent on software (all other concerns aside) are the hand-marked, hand-counted paper ballots and the hand-marked ballots with cryptographic verification.

If one chooses to exclude the systems with hand-marked ballots (shaded in grey) from consideration, due to the potential usability, accessibility, and accuracy advantages of computer-based vote entry, then the bottom two options in the "public-ballot electronic" category are the least dependent on software. A system based on a DRE with a published ballot definition and published vote records will use the least amount of critical software, but also requires voters to place great trust in that software. A system based on an EBP with a published ballot definition will be dependent on the optical scanner's software as well as the EBP software, but both software-dependent steps are subject to paper-based checks. The choice between these two options would depend on one's confidence in the ability to verify DRE software and one's estimate of the likelihood that significant errors will be caught by observant voters and recounts.

All of the systems that involve entry of votes using any kind of voting computer—DRE, EBM, or EBP—could stand to benefit from easier verification of the software in that computer. This is where we will turn our attention in the next chapter.

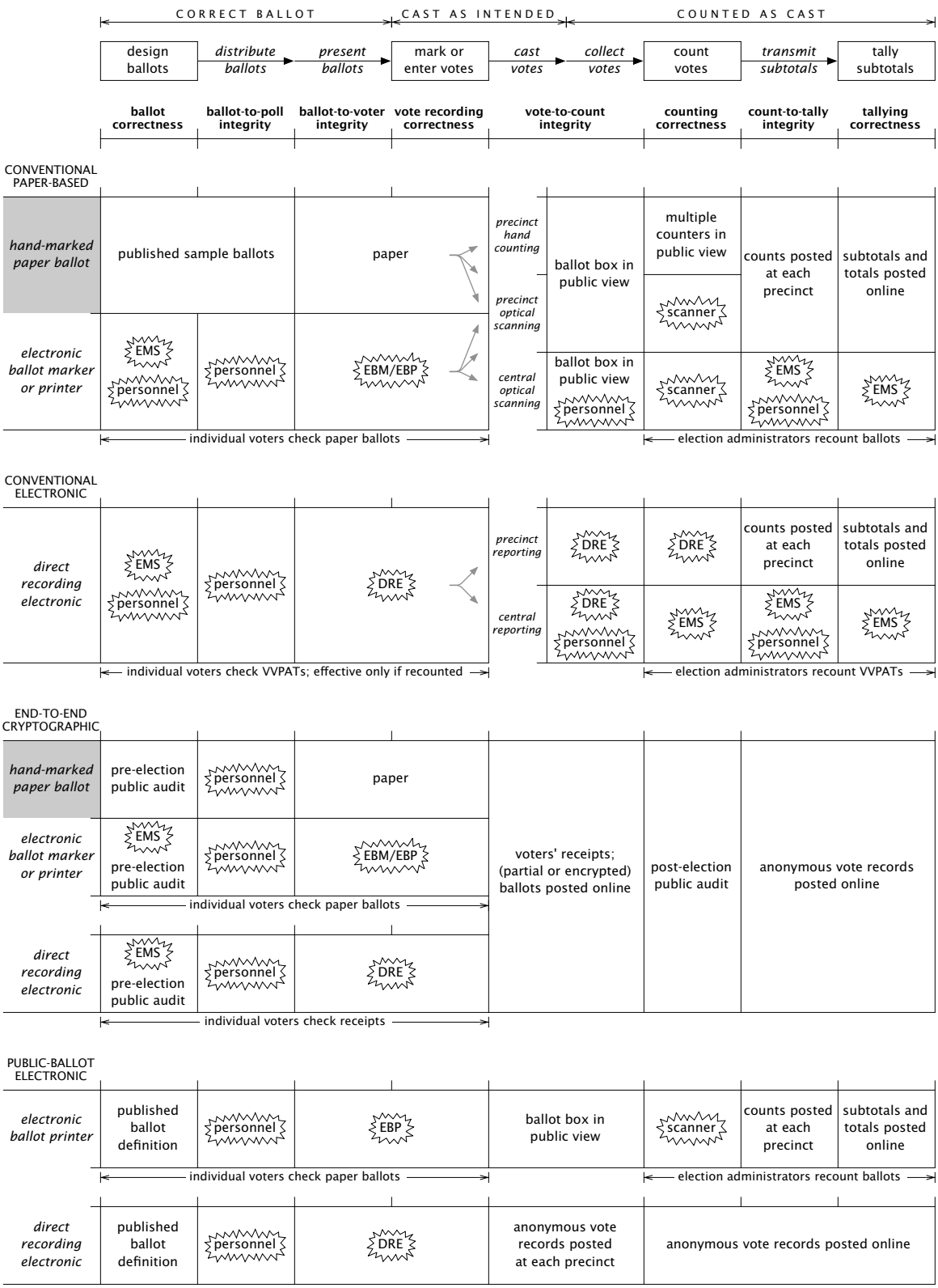


Figure 3.9. Summary of assurance mechanisms for various types of voting systems.