On the notion of "software independence" in voting systems

Ronald L. Rivest
John P. Wack

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
rivest@mit.edu

National Institute of Standards and Technology (NIST)
Information Technology Laboratory
Gaithersburg, MD 20899
john.wack@nist.gov

DRAFT Version July 28, 2006

Abstract. This paper defines and explores the notion of "software independence" in voting systems:

> A voting system is *software-independent* if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome.

We propose that software independent voting systems should be preferred, and *software-dependent* voting systems should be avoided. VVPAT and some cryptographically-based voting systems are software-independent. Variations and implications of this definition are explored.

This white paper is also for discussion by the Technical Guidelines Development Committee (TGDC) in its development of the Voluntary Voting System Guidelines (VVSG) of 2007.

1 Introduction

The main purpose of this paper is to introduce and carefully define the terminology of "software-independent" and "software-dependent" voting systems, and to discuss their properties. This paper is definitional in character; there are no "results" as such. The contribution is to help provide crisp terminology for discussing voting systems and some recommendations for its usage in the VVSG 2007, currently under development by NIST and the TGDC.

This paper starts by describing the problem that software-independence addresses: complex and difficult-to-test software in voting systems. It then defines what constitutes a software-independent approach to voting system design. The

paper provides examples of this approach and then discusses various issues, including ramifications to testing and issues with usability and accessibility. It concludes with recommendations for the role of software-independence in the VVSG 2007.

This working note is intended to stimulate discussion, and does not represent an official document or any official position of NIST, EAC, TGDC, or MIT.

2 Problem: Software complexity of voting systems

Electronic voting systems are complex and getting more so, as both elections themselves and voter interfaces get more complex. The requirements for a voting system are also very demanding: the requirements for accuracy of the final tally, privacy of individual votes, and security against attack are in serious conflict with each other. Conflicting requirements also usually leads to burgeoning system complexity.

As a consequence, voting systems express and capture this complexity via software; software provides a powerful means of describing the complex patterns of behavior that a voting system must exhibit. Perhaps the best example of this is the Direct-Recording Electronic (DRE) voting system, which typically provides a touch-screen user interface for voters to make selections and cast ballots, and stores the cast vote records in memory and on a removable memory card. A DRE may display and use potentially thousands of different ballot layouts. A DRE may also include complex accessibility features for the sight-impaired such that a voter could use headphones and be guided to make selections.

An issue, then, is how to determine, despite the complexity of the software, whether the voting system is accurately recording the voters intentions. The DRE voting system produces only one instance of its cast ballot records (there is no second independently-created set of records for which to compare them), consequently the accuracy of the records must be ascertained by a variety of (imperfect) measures. These include comparing the accumulated tallies to pre-election canvassing results as a measure of their expected accuracy, and techniques such as parallel testing to gauge voting system accuracy.

Fundamentally, though, one must trust that the software was written and tested well, that the software running on the system is indeed the certified, tested software, and that no tampering of the software has occurred.

2.1 The difficulty of evaluating complex software for errors

However, it is a common maxim that complexity is the enemy of security—it is very difficult to evaluate the security of a complex system. A very small error, such as a transposed pair of characters or an omitted command to initialize a

variable, in a large complex system may provide a vulnerability that can be exploited by an adversary for large benefits. Or, it may simple cause unexpected results at unpredictable intervals.

Finding all errors in a large system is generally held to be impossible in general or else highly demanding and extremely expensive. Our ability to develop complex software vastly exceeds our ability to prove its correctness or test it satisfactorily within reasonable fiscal constraints (extensive testing of a voting system's software would certainly be cost-prohibitive). A voting system for which the integrity of the election results depends on the correctness of its software will always be somewhat suspect and require routine checks of its software, even after extensive (and expensive) federal testing and certification.

2.2 The need for software independent approaches

One should strongly prefer any approach where the integrity of the election outcome is not dependent on trusting the correctness of complex software. Voter-verified paper audit trails (VVPAT) provide the most prominent (albeit ad hoc) approach available today in the market. But there are other approaches possible, such as those based on novel cryptographic techniques that promise levels of assurance of correct election outcomes that exceed those provided by simple voter verifiable paper audit trails.

What does this mean for voting systems? The purpose of this paper is to provide a new notion, that of "software independence," that captures the essence of the problem.

Voting systems that are "software dependent" rely on the correctness and integrity of their software in ways that "software independent" systems do not. The complexity of the software in "software-independent" voting systems is much less of a problem.

Software-independent voting systems should support much greater assurance of the correctness of their election outcomes; there is no lingering unanswerable concern that the election outcome was actually determined by some software bug or worse (e.g., a malicious piece of code).

3 Definition and rationale for software independence

We now repeat the definition of software independence, and explore its meaning.

> A voting system is *software-independent* if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome.

A voting system that is not software-independent is said to be "software-dependent"–it is, in some sense, vulnerable to undetected programming errors or malicious code; the correctness of the election results are dependent on the correctness of the software and on whatever assurances can be obtained that the software on the voting machine is in fact the software that is supposed to be there.

These notions are not exactly new—many have discussed the problems associated with using complex software in voting systems. Yet, we have lacked crisp terminology for talking about the dependence of election outcomes on such complex software.

3.1 Refinements and elaborations of software independence

There are a number of possible refinements and elaborations of the notion of software independence. We now motivate and introduce the distinction between *strong software-independence* and *weak software-independence*.

Security mechanisms are typically one of two forms: *prevention* or *detection*. Detection mechanisms may also be coupled with means for *recovery*. When identification of participants and accountability for actions is also present, then detection mechanisms are also the foundation for *deterrence*.

In voting systems, *preventing* software changes and errors is very difficult, given the difficulty of assuring software correctness, our current level of investment in voting system security, the distributed and infrequent nature of elections, and the volunteer status of many election workers. Trying to justify the adoption of software-dependent voting systems on the basis that software changes and errors can be entirely prevented seems very unrealistic. So, relying as well on strong *detection* methods is well motivated.

Given the importance of recovery mechanisms in addition to detection mechanisms, we propose the following two definitions:

> A voting system is *strongly software-independent* if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome, and moreover, a detected change or error in an election outcome (due to change or error in the software) can be corrected without re-running the election.

A voting system that is *weakly software-independent* conforms to the basic definition of software-independence, that is, there is no recovery mechanism.

As an illustration of these terms, consider the DRE. Even if used with parallel testing, it is not software-independent. If it produces a voter-verifiable paper audit trail, it is software-independent. If its paper trail meets all requirements

in a given election for use as an official ballot of record, it is strongly software-independent - if errors are detected, the paper trail can be legally used and the election need not be re-run. If its paper trail is useful only for showing errors, it is weakly software-independent, i.e., it meets the basic requirements for software-independence.

3.2 Usability of software-independent approaches

A software-independent approach is able to detect errors via voter-verification (if used) and via audits of its cast ballot records. It is possibly able to recover from errors and problems depending on the suitability of its records for recounts. Therefore, it follows that usability issues affecting voter-verification and cast ballot record production are paramount in consideration of software-independent approaches. This paper does not address or detail those issues further other than to note that these issues must be considered as a fundamental to the design of voter-independent approaches. Otherwise, if voter-verification or audits are complex or difficult to use, software-independence quickly becomes software-dependence.

3.3 Examples

In general, voting systems that have a voter-verifiable paper audit trail are software-independent, since the paper audit trail allows (via a recount) the possibility of detecting (and even correcting) errors due to software. Accordingly, these voting systems can be strongly software-independent.

In this category we should include not only DRE voting systems that have been augmented with VVPAT, but also electronic ballot marking systems (EBMs) and mark-sense (optical scanning) systems. This approach has the advantage of isolating the complex user-interface software on the EBM from the more critical ballot recording and counting software on the optical scanner. The EBM prints a high-quality paper ballot of the voter's choices, which can be verified for accuracy before being fed into the optical scanner; the paper ballot serves as a paper trail that can be used in audits of the optical scanner's records. This usage of voting devices provides both detection and recovery, and is strongly software-independent.

Cryptographic voting systems are another example of software-independent voting systems. They can provide detection mechanisms for errors caused by software changes or errors (e.g. [4,5,9,10,3]). At one level, they can enable voters to detect when their votes have been improperly represented to them at the polling site, and a simple recovery mechanism (re-voting) is available. At another level, they can enable anyone to detect when the official tally has been computed incorrectly. Recovery is again possible, assuming that the tally administrators

still possess the necessary cryptographic key information. Most of the recently proposed cryptographic voting systems are strongly software-independent.

(We note that in many of the cryptographic schemes the detection of vote mis-representation is probabilistic; the voter can "catch" the misbehavior of a voting system with probability at least 1/2. The ability of a voting system to undetectably mis-represent more than a few votes becomes vanishingly small very quickly. If one wanted a term to distinguish these schemes from schemes (such as op-scan) where the ability of the voter to catch mis-representations was guaranteed, one might call the cryptographic schemes "*virtually software-independent*, whereas a VVPAT or op-scan is "*strictly software-independent*".)

4 Relationship to Independent Verification in the VVSG 2007

The terms *Independent Verification (IV)* and *Independent Dual Verification (IDV)* have been used by NIST and the TGDC to describe voting systems that produce multiple cast ballot records, at least one of which is immutable and can be verified by the voter to be correct. IV/IDV was included in the VVSG 2005 as informative text, and NIST and the TGDC have been considering requiring that voting systems in the VVSG of 2007 meet its requirements. Its essential requirements are:

1. At least two records of the voter's choices are produced and one of the records is then stored such that it cannot be modified by the voting system, e.g. the voting system creates a record of the voters choices and then copies it to some write-once media (e.g., paper).
2. The voter must verify that both records are correct, e.g., verify his or her choices on a DREs display and also verify the second record of choices stored on the write-once media.
3. The verification processes for the two verifications must be independent of each other and (a) at least one of the records must be verified directly by the voter, or (b) it is acceptable for the voter to indirectly verify both records if they are stored on different systems produced by different vendors.
4. The content of the two records can be checked later for consistency.

VVPAT is the most obvious example of a voting system that provides IV or IDV, and other approaches have been discussed that have the potential of using other media besides paper. IV/IDV remains an important concept for the VVSG 2007, as it describes how records must be produced in certain types of voting systems so that they can be said to be software-independent.

The third requirement for IV/IDV opens the possibility of two independent verifications being permissible. As an example, a DRE could be attached to a second system and thus transmit its electronic cast vote record to that system

after a voter has indicated they have completed the ballot. However, this is an apparent violation of the software-independence approach because the verifications are both dependent on the accuracy of the software. We would assert that, practically speaking, enforcing a rule requiring that different vendors produce the systems would be difficult at best and not likely to counter the software-dependent approaches of both systems. Therefore, we recommend that part (b) of this requirement be dropped.

As a primary concept for use in the VVSG 2007, ID/IDV misses the mark in that it describes a technique to achieve software-independence but does not focus on the problem it is attempting to address, that being the inability to verify complex software in voting systems. Consequently, arguments for or against it have focused more on issues concerning voter-verification of paper records, e.g., the additional cost of VVPAT systems and the usefulness of the paper records in audits. We assert that the term *software-dependence* better focuses the argument on the difficulty and expense of evaluating complex code and subsequently trusting that it doesn't contain errors or that the voting system software has not been tampered with.

5 Are parallel testing and other measures sufficient for software-dependent approaches

Parallel testing [8] is often cited as an efficient and accurate gauge of the correct operation of a voting system and, by implication, the correctness of its software. However, this approach is designed to detect software changes or errors, not to detect whether the election outcome has been affected by such changes or errors. A problem detected during parallel testing may or may not indicate an actual problem during the election. The best one can do when parallel testing uncovers a problem is assume the worst. There is no obvious recovery mechanism available, other than re-running the election.

Furthermore, reliance on parallel testing to detect errors would require that the testing be done in a very comprehensive manner for each use of the voting system, such that the voting systems capabilities are thoroughly exploited and tested. Determining how extensive the tests should be could itself be quite complicated and labor intensive. For example, a voting system expected to hold 10,000 votes but in reality holding only 3,000 [7] likely would not have been detected by parallel testing, yet this incident resulted in 4,400 lost votes and an election partially re-run. Given the large number of voting jurisdictions with varying procedures, varying levels of expertise with electronic voting systems, and a largely volunteer force of poll workers, it seems more likely that parallel testing can be at best an approximate or possibly rough gauge of software accuracy.

Logic and accuracy (L&A) testing is also cited as an effective detection method, but again it suffers from the same quality deficiencies as parallel testing.

Furthermore, some voting systems actually perform logic and accuracy testing on a separate base of software and do not test the operational voting system software used to conduct elections.

Additionally, the VVSG 2005 contains requirements for (a) more secure and verifiable mechanisms and procedures for distributing certified voting system software, and (b) voting systems to support secure election-day validation of their software (setup validation). The purpose of these requirements is to assist in ensuring that the correct certified software is shipped with the voting system and that the correct certified software is actually running on the voting system. These requirements are sometimes confused as being sufficient for ensuring that a software-dependent approach is using correct, error-free software; they ensure only that the system is using the correct *certified* (lab tested) software. These requirements are highly recommended regardless, but alone or in combination with techniques such as parallel or logic and accuracy testing, they offer no guarantees that the correct, certified software is actually error-free.

6 Discussion

6.1 Implications for testing and certification

Given the exceptional difficulty of proving software to be correct, and given the difficulties of maintaining tight physical control over a multitude of voting machines (so as to prevent tampering with software), it is a reasonable proposal to disallow voting systems that are software-dependent altogether.

If testing and certification of software-dependent voting systems are to be nonetheless contemplated, then one should reasonably expect the certification process should be very much more demanding and rigorous for a software-dependent voting system than for a software-independent voting system. The manufacturer of a software-dependent voting system should submit, as part of the evaluation package, a formal proof of correctness. Perhaps an assurance level corresponding to EAL level 6 or 7 should be required, whereas for a software-independent system [2,1] an lower assurance level (or the equivalent) would be the norm. It is reasonable to expect that when the correctness of election outcomes depends fundamentally on the correctness of software, that the software producer should have to work significantly harder to assure potential customers of the correctness and security of its system. Moreover, the potential customer needs to have rigorous procedures in place to assure that the system utilized during an election is indeed the same as what was evaluated and purchased.

6.2 Disabled voters

Since a blind voter may be unable to verify a printed ballot without assistance, a voting system that is software-independent for typical voters may not be

software-independent for a blind voter; the blind voter may use software support to read back his or her choices.

We propose that the notion of "software-independence" be understood (unless further qualification is given) to refer to the qualities of the voting system for typical voters; it being understood that the system may be qualitatively different for a disabled voter. This difference may be unavoidable, but is worth noting when considering each system.

That said, some electronic ballot marking devices and cryptographic voting systems that use a DRE-like interface carry great promise for accessibility. Furthermore, accessibility-related software is expected to be complex, thus sticking as much as possible to software-independent approaches offers greater potential for accessibility features to be included in voting systems without necessarily requiring expansive and highly-expensive testing.

6.3 Interoperability issues

Software-independent approaches rely on audits of cast ballot records to detect errors and problems. Requiring a common, interoperable format for electronic representations of cast ballot records would assist in comparisons as well as tabulations, especially when combining the output of different types of voting systems (e.g., combining and tallying records from an accessible voting station and a VVPAT system). OASIS Election Markup Language (EML) [6] or a scaled-down variant is a likely choice.

6.4 Transparency

The issue of transparency is important in that voters should be able to understand in general how the voting system "works." This is important to any voting system approach but may be more so to software-independent cryptographic approaches, which can be difficult to understand and therefore not especially transparent.

At the same time, voters do not need to understand complex cryptographic methods and protocols to have an adequate comprehension of the voting system and level of comfort in using it — if the design is somewhat simple to grasp, voters could, with the passage of some time, be comfortable using systems that may be very complicated "under the hood." Consider, for example, how readily people now use and trust web-based transactions underpinned by SSL (e.g., *https://...*). Simply adding an "s" at the end of "http" and associating it with "secure" seems to be satisfactory.

6.5 Extensions and variations

The notion of "software-independence" speaks only to the goal of accuracy (correctness of election outcome). Thus, we might have termed this notion "*software independence for correctness*".

Using similar qualifiers, we can also create and consider related notions, such as "*software-independence for voter privacy*".

And of course, we could think of independence of other parts of the voting system, "*hardware-independence for correctness*" or "*poll worker-independence for voter privacy*".

7 Conclusions and suggestions

We have suggested the use of the terms "software-independence" and "software-dependence" to describe whether or not the correctness of election results depends in an essential way on the correctness of voting system software.

Should software-independence be mandated in the VVSG 2007? The history of computing systems is that, given improvements and breakthroughs in technology and speed, software is able to do more and thus its complexity increases. The ability to prove the correctness of software diminishes rapidly as the software becomes more complex. It would effectively be impossible to adequately test future (and current) voting systems for flaws and introduced fraud, and thus these systems would always remain suspect in their ability to provide secure and accurate elections. The cost of effective testing would be prohibitive and could place restraints on vendors to introducing new (and software-intensive) improvements in voting technology.

Adopting the software-independent approach would place fewer restraints on the market place to develop new and improved technology for voting systems. As long as the validity of election results does not fundamentally depend on software correctness, vendors may better address increasing usability and accessibility needs in an aging and increasingly diverse population. It should be noted that software independence will not obviate the need for strong and thorough testing, in fact testing of future voting systems may well be more expensive than today.

References

1. Common criteria assurance levels. Available at: `http://www.cesg.gov.uk/site/iacs/index.cfm?menuSelected=1&displayPage=13`.
2. Common criteria evaluation and validation scheme. Available at: `http://niap.bahialab.com/cc-scheme/`.

3. Ben Adida. *Verifying Secret-Ballot Elections With Cryptography*. PhD thesis, MIT Department of EECS, August 2006.
4. David Chaum. Secret ballot receipts: True voter-verifiable elections. *IEEE J. Security and Privacy*, pages 38 – 47, Jan/Feb 2004.
5. David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical, voter-verifiable election scheme. Technical Report CS-TR-880, University of Newcastle upon Tyne School of Computing Science, December 2004. Available at: `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/880.pdf`.
6. Organization for the Advancement of Structured Information Standards. Oasis election markup language specification. Available at: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=election`.
7. Heather Havenstein. Voting system ballot counter overflow. *Computerworld Magazine*, Dec 2004. Available at: `http://www.computerworld.com/governmenttopics/government/story/0,10801,98054,00.html`.
8. Douglas Jones. Parallel testing during an election. Available at: `http://www.cs.uiowa.edu/~jones/voting/testing.shtml#parallel`.
9. Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: A system perspective. In *Proceedings 14th USENIX Security Symposium*, August 2005. Available at: `http://www.cs.berkeley.edu/~nks/papers/cryptovoting-usenix05.pdf`.
10. C. Andrew Neff. Practical high intent verification fo encrypted votes, October 2004. Available from VoteHere.
11. Peter Y. A. Ryan and Thea Peacock. Prêt à Voter: A system perspective. Technical Report CS-TR-929, University of Newcastle upon Tyne School of Computing Science, September 2005. Available at: `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/929.pdf`.
12. Peter Y. A. Ryan and Steve A. Schneider. Prêt à Voter with re-encryption mixes. Technical Report CS-TR-956, University of Newcastle upon Tyne School of Computing Science, April 2006. Available at: `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/956.pdf`.