Volume 84, issue 10, October 2011
ISSN 0164-1212
ELSEVIER
The Journal of
Systems and Software

The Journal of Systems and Software 84 (2011) 1618–1637

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Formal analysis of an electronic voting system: An experience report

Komminist Weldemariam[a,*], Richard A. Kemmerer[b], Adolfo Villafiorita[a]

[a] Foundation Bruno Kessler, via Sommarive 18, TN 38123 Trento, Italy
[b] Department of Computer Science, University of California, Santa Barbara, CA 93106-5110, United States

ARTICLE INFO

Article history:
Received 22 May 2010
Received in revised form 16 January 2011
Accepted 11 March 2011
Available online 23 March 2011

Keywords:
Electronic voting systems
ES&S system
Formal specification and verification
Critical requirements

ABSTRACT

We have seen that several currently deployed e-voting systems share critical failures in their design and implementation that render their technical and procedural controls insufficient to guarantee trustworthy voting. The application of formal methods would greatly help to better address problems associated with assurance against requirements and standards. More specifically, it would help to thoroughly specify and analyze the underlying assumptions and security specific properties, and it would improve the trustworthiness of the final systems. In this article, we show how such techniques can be used to model and reason about the security of one of the currently deployed e-voting systems in the U.S.A named ES&S. We used the ASTRAL language to specify the voting process of ES&S machines and the critical security requirements for the system. Proof obligations that verify that the specified system meets the critical requirements were automatically generated by the ASTRAL Software Development Environment (SDE). The PVS interactive theorem prover was then used to apply the appropriate proof strategies and discharge the proof obligations. We also believe that besides analyzing the system against its requirements, it is equally important to perform an analysis under malicious circumstances where the execution model is augmented with attack behaviors. Thus, we extend the formal specification of the system by specifying attacks that have been shown to successfully compromise the system, and we then repeat the formal verification. This is helpful in detecting missing requirements or unwarranted assumptions about the specification of the system. In addition, this allows one to sketch countermeasure strategies to be used when the system behaves differently than it should and to build confidence about the system under development. Finally, we acknowledge the main problem that arises in e-voting system specification and verification: modeling attacks is very difficult because the different types of attack often cut across the structure of the original behavior models, thus making (incremental or compositional) verification very difficult.

1. Introduction

Electronic voting (e-voting) brings to the polling station several advantages, such as improved turn out, accessibility for impaired people, and improved accuracy and speed (Cranor, 1996). Unfortunately, its adoption in various countries has been slow and/or the cause of debate and controversies. One of the reasons is the poor design and implementation of (some of) the systems currently deployed for elections in the USA and other countries, as different studies have reported and demonstrated (Kohno et al., 2004; Aviv et al., 2008; Balzarotti et al., 2010; Wolchok et al., 2010). These studies have also revealed that such systems show serious flaws in specification, design, and implementation. Such weaknesses expose the system, and consequently elections, to various threats and attacks, ranging from a denial of service to alteration of the results.

In California, these studies resulted in the Secretary of State allowing the use of e-voting machines only in special situations and with various changes to the electoral procedures. Several such changes shift the implementation of security requirements from e-voting systems to poll workers. For instance, California Secretary of State (2007) states that *“no poll worker or other person may record the time at which or the order in which voters vote in a polling place.”* It is quite evident that a new generation of more carefully engineered machines could move various “constraints” currently performed by poll workers back to hardware and software. However, the success of the new generation of voting machines depends on our ability to capitalize on the lessons we learned using and analyzing the systems currently deployed.

The integrity and assurance of a complex and safety-critical system’s correct behavior with respect to a specification can be achieved if good engineering practices are appropriately devised and used. With respect to this, there are a number of approaches to tackle (some of) the issues mentioned above. Among these, the use of formal methods has been shown to improve the security and quality of complex systems (Kemmerer, 1990; Xu and Nygard,

* Corresponding author.
E-mail addresses: sisai@fbk.eu, komminist@gmail.com (K. Weldemariam).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.03.032

2005; Lowry and Dvorak, 1998; Heitmeyer et al., 2008). Formal techniques allow designers to prove, test, or otherwise examine interesting properties of a complex process whose behavior is specified abstractly, and then interactively refine the behavioral specification to be as close to an implementation as appropriate for a given assurance level.

The use of formal methods in the voting domain is still at an early stage. Some of the works describe and demonstrate the feasibility of using formal methods on specific components, such as the cryptographic protocols used to protect and transmit data (see e.g., Juels et al., 2005; Kremer and Ryan, 2005; Campanelli et al., 2008; Delaune et al., 2009). Others focus on the verification of general properties of e-voting systems (see e.g., Simidchieva et al., 2008; Villafiorita et al., 2009; Sturton et al., 2009). Even though all the works mentioned earlier provided a significant contribution to the area, they are limited in scope or refer to schemas that do not find application in machines currently in use. In this article, our specific focus is on the systematic use of formal methods to study and analyze the strength and weaknesses of currently deployed e-voting machines in the USA. We did so, by deriving formal specifications along with critical security requirements for the Election Systems & Software (ES&S) system.

More specifically, we treated the ES&S voting system as a complex, real-time embedded system, consisting of a direct recording election machine (DRE), a real-time audit log printer (RTAL), a personalized election ballot (PEB), and a Compact Flash Card (CF). We mapped each of these components to ASTRAL (Kolano, 1999) process instances. We then specified critical security requirements to prove the correctness and integrity of each component individually and of the system as a whole. The consistency of the specification was validated using the ASTRAL validation engine, and PVS (Owre et al., 1993) proof obligations were automatically generated by the ASTRAL Software Development Environment (SDE). When proved, these proof obligations verify that the specified system meets the critical security requirements. The PVS interactive theorem prover was used to apply the appropriate proof strategies and discharge each of the proof obligations. Additionally, we specified attacks that have been shown to successfully compromise the system. With this information, we extend the original specification of the system and derive what we called the extended model. Using the same machinery, we reason that the same critical requirements do indeed hold in the extended model. The techniques presented in Kolano (1999) were extensively used to discharge the proof commands. We must also be clear that we did not complete all the proof obligations. By analyzing the proved obligations, however, we attempted to understand why some were proved and why the others were not.

This article is organized as follows. Section 2 discusses the motivation for the work. Section 3 presents the various components of the ES&S system, the voting process using the system, and requirements that the system must respect. Moreover, we present four selected attacks. An overview of the ASTRAL specification language is given in Section 4. The ASTRAL specification of the ES&S system along with its critical requirements are presented in Section 5. Sections 6 and 7 present the specification of the attack scenarios – namely, the model extension and the verification results, both before and after the attack specifications. Finally, Section 8 discusses related work and Section 9 draws some conclusions.

2. Motivation and the approach

The fairness and security of electronic elections depend upon a careful allocation of requirements to the procedures and to the systems used. In fact, the correct behavior of the electronic systems can be guaranteed when they are used according to their operating specifications. This has to be guaranteed by the procedures and the people responsible for executing them. It would be possible to imagine an e-voting machine that uses a specific technique (e.g., biometrics for voter authentication) to identify a voter and prohibit the casting of a vote from someone who has already voted. However, given the procedures and systems that are currently in use, there is no way for an e-voting machine to prohibit the same person from casting multiple ballots, if the poll worker enables the machine for voting to the same person multiple times.

As a matter of fact, in all DRE systems studied the poll worker uses an administrative device to issue a token of some sort for an eligible voter to cast a vote (see in Aviv et al., 2008; Inc, 2007). Such behavior can possibly be prevented (or revealed after the election) by enforcing and verifying the procedures that the poll workers are supposed to follow. In contrast, there are other fundamental properties that the procedures can only partially assure. In this case, the e-voting systems must guarantee that these properties are satisfied. Using the example we just made, the machine must ensure that a voter can cast at most one vote, given that the poll workers follow the prescribed procedures.

Formal analysis of voting requirements and their allocation is therefore important in two aspects. First, it helps to ensure that the systems meet the necessary reliability and dependability goals. Second, it helps to better understand how different allocations of requirements between systems and procedures could improve the overall security of the election process so that we can build the next generation of e-voting machines. However, in order to achieve these goals, we need an approach that allows us to easily experiment and reason about, for example, different allocations of requirements. At the same time, it has to be precise and exhaustive, so that security and dependability consequences of any specific choice are highlighted. Formal techniques clearly fit both needs. Our goal here is not to show end-to-end verification nor to develop an e-voting system using formal methods; instead, we wish to demonstrate how their use can help ensure fair elections.

Fig. 1 depicts the reverse synthesis process that we use. In the figure, the *nominal* behavior refers to all the intended operations of the system under analysis. By contrast, the *non-nominal* behavior is meant to describe those behaviors of the system that deviate from intended operations of the system due to attack actions. More specifically, we derive formal specifications along with critical security requirements for the ES&S system. The specification of the system (i.e., *Model* in Fig. 1) and the security critical requirements are mainly derived from available information sources: the EVEREST report (McDaniel et al., 2007), the ES&S election day checklist and user's manual (Inc, 2007), a video (Election and Systems, 2009) that shows how the ES&S system works on election day, and other requirements suggested in the literature such as Mercuri (2001), Federal Election Commission (2005), Council of Europe (2004), Sastry (2007), Volkamer and McGaley (2007), and McGaley (2008). Here, we remark that the choice of selecting requirements from the cited literature is made based on our experiences managing and structuring requirements while developing the ProVotE e-voting system (Weldemariam et al., 2009). Using formal analysis tools, we can assess the strengths and weaknesses of the system. Results or feedback gained from the formal analysis can be a basis for specifying and analyzing generic requirements from which the next generation of voting machines can be built.

Additionally, we believe that besides analyzing the system against its requirements, it is equally important to perform analysis under malicious circumstances where the system execution model is enriched with attack behavior. Notice that the first model we build for the ES&S system only specifies the intended behaviors of the system. Therefore, we should specify and extend the model (see Fig. 1) with some generic attacks that may defeat some behaviors of the system. We then need to analyze the resulting model (i.e., *extended model* in Fig. 1) against the same security properties that are used for the verification of the nominal model.

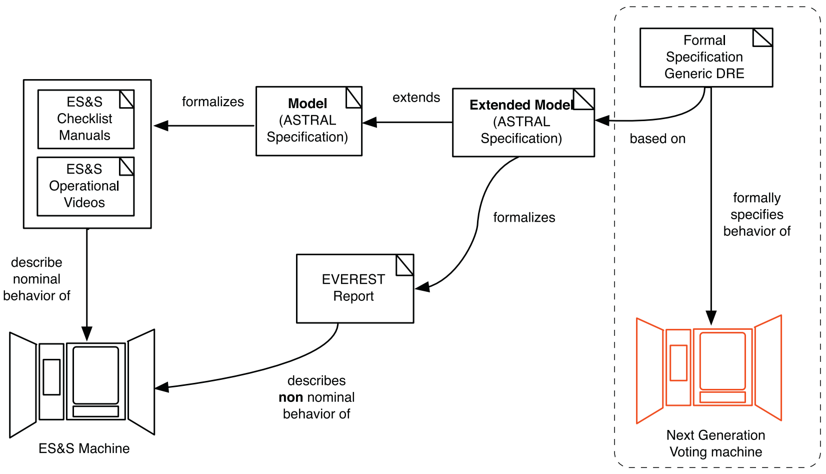

ES&S Checklist Manuals
ES&S Operational Videos
formalizes
Model (ASTRAL Specification)
extends
Extended Model (ASTRAL Specification)
Formal Specification Generic DRE
based on
formally specifies behavior of
formalizes
describe nominal behavior of
EVEREST Report
describes non nominal behavior of
ES&S Machine
Next Generation Voting machine

Fig. 1. Overview of our approach. Labels nominal and non nominal refer to the actual and undesired behaviors of the system, respectively.

A successful analysis of the resulting model can reveal important information about the system, which in turn helps detect missing requirements or unwarranted assumptions about the specifications that we developed. In addition, this allows us to sketch countermeasure strategies to be used when the system behaves differently than it should and to build confidence about the system under development.

Finally, we must be clear that the formal specifications of a generic DRE (see the part in the dotted box of Fig. 1), where the behavior of the next generation voting machine can be specified, is not in the scope of this paper. However, we believe that starting from our specification and the proof results, one can specify (and analyze) requirements for new e-voting systems, mainly for DRE-based systems.

3. The ES&S electronic voting systems

In this section, we first describe the ES&S voting system components and then (informally) a set of security critical requirements for these components individually, as well as for the system as a whole. The formal specifications are based on this information.

3.1. ES&S voting system components

Our discussion of the ES&S voting system components is based on what each component does, how each component exchanges input or output, and the underlying assumptions made about each component. For the purposes of this work, the ES&S voting system is composed of:

- *DRE*: Direct Recording Electronic voting machine, called the iVotronic. It is equipped with a touch-screen where the voter casts his/her votes. The information shown by the touch-screen changes in real-time to match the voter's choices. The iVotronic also stores the audit data.
- *RTAL*: Real-Time Audit Log Printer, which performs the function of a VVPAT (Voter-Verified Paper Audit Trail) for the ES&S system. It produces a paper-based record of the choices selected by the voter. The RTAL is plugged into the DRE and the paper record is viewable by the voter. The trails (i.e., the voter's choices) are under a transparent cover so that they cannot be modified other than through the normal voting process.
- *PEB*: Personalized Electronic Ballot. This is a device used by the poll worker to load a ballot, initialize the next ballot, and collect tabulated data and audit information. Each time a PEB is inserted, its authenticity is checked by the DRE using a four-digit code (election qualification code, EQC), which is assigned prior to election day.
- *CFC*: Compact Flash Card. This device holds files too large to fit in the PEB and also audit data. The card must be present to open and close the polls. At poll closing, the audit data is automatically dumped into the card.

The interaction between these components is as follows (see also Fig. 6). The DRE communicates with the RTAL by sending the voter's intentions and information related to the casting of a ballot, such as the start or summary information. The PEB communicates with the DRE through a simple protocol that allows the DRE to read and write memory blocks stored in the PEB (e.g., to load ballots before election, or to enable the ballot when an eligible voter arrives). Similarly, the DRE communicates with the CF Card to access the ballot data when necessary and to periodically check the presence of the CF Card, since the DRE will not boot without its presence (see McDaniel et al., 2007 for a more detailed and complete view).

3.2. Voting process for a DRE based system

The full election process involves many activities beyond what a poll worker and a voter typically experience in the polling station. Even if the exact processes differ depending on the specific voting technology in question, we distinguish, in particular, three major phases in the voting process when using DRE-based machines: pre-electoral, electoral (during voting), and post-electoral phases. Before election day, election officials use the election management system (EMS) to set up the election. In particular, the ballot definition files are prepared and loaded directly onto the DREs, CF Cards are installed, and printers are assigned for each DRE machine. Moreover, the EQC is stored in the DRE so that the DRE can authenticate a qualified PEB when one is inserted.

Prior to opening the polls, a poll worker unpacks and sets up the DRE and plugs in the RTAL printer and power cables. Poll workers must also ensure that a properly programmed CF Card is installed before powering on the DRE. A Master PEB is inserted into the terminal to load the ballot and later to open the DRE terminal for voting. The same master PEB must be used to close the terminal after the polls have closed. Removing the PEB turns the terminal's current mode to sleep mode.

Once the polls are opened, a poll worker initializes the ballot for a qualified voter by inserting a supervisor PEB, which can be the same Master PEB used to open the polls, into the machine. The terminal mode changes from sleep to poll worker mode, the EQC code of the PEB is checked, and the ballot is initialized, provided that

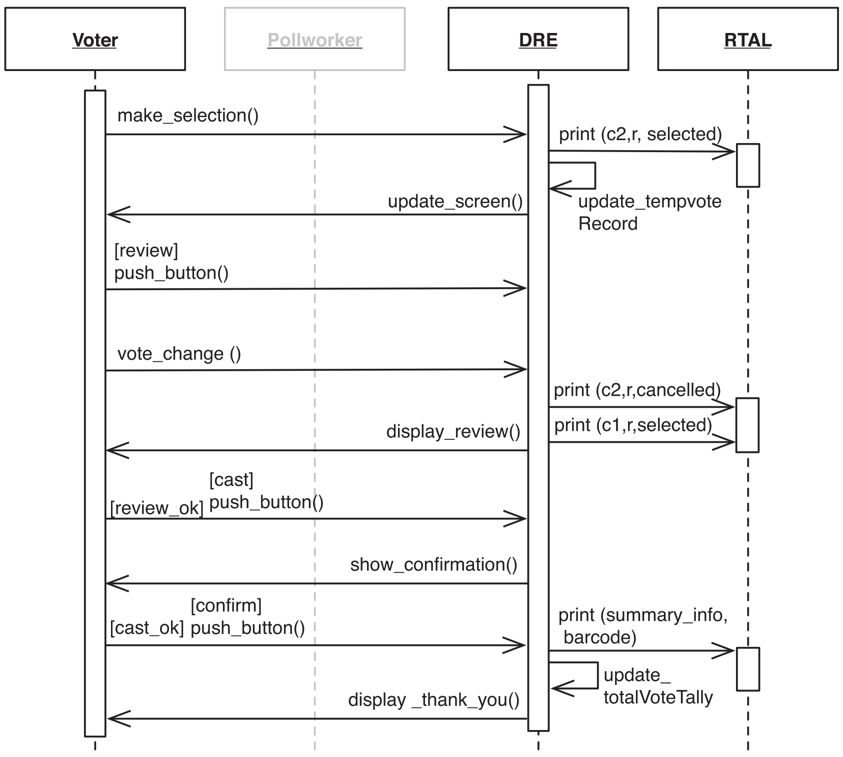


Fig. 2. The voting process using the ES&S voting system. The figure shows only parts of the process after the poll worker activated the machine for an eligible voter.

the EQC of the PEB matches with the one the terminal is configured for. The poll worker removes the supervisor PEB and leaves the terminal for the voter.

After the ballot is activated, the machine takes the voter through each contest. The ES&S DRE machines automatically forbid overvoting, but not undervoting. When a voter selects or cancels a candidate for a particular contest, an appropriate indication is printed on the RTAL record. If the voter selects a candidate, the RTAL record is marked as "Selected" and scrolled out of sight; otherwise, it is marked as "Canceled" and scrolled out of sight. The voter is eventually given the opportunity to review his ballot, and if the voter commits to it (confirms it), it is recorded to local storage. The process continues in this way for all qualified voters (see Fig. 2).

After the official poll closing time is reached and there is no qualified voter waiting in line, the poll worker inserts the master PEB to collect and store tabulated data, copies of the ballot image (i.e., file) and some other information. Upon closing the terminal, the DRE firmware automatically uploads the audit data onto the CF Card. The results tape from the RTAL is also collected. The results tape, CF Card, and master PEB from each polling place are then returned to election central.

We remark that some properties documented in the ES&S election day checklist manual – such as, "while downloading the election results from the DRE after the election is closed, the PEB should not be removed until the download finishes and it is safe to remove it" – are not intrinsic to the system functionality. They are either procedural and/or environmental assumptions.

3.3. Informal requirements for the ES&S system

We describe a list of security properties that the system must respect. The security goal is that even in the presence of an adversary, the system should meet these properties. For instance, the DRE should record the voter's intent exactly as the voter desires. Furthermore, an adversary should not be able to undetectably alter the votes once they have been successfully stored. We wish to specify these kinds of properties and validate against the system model, as well as in the presence of threat actions corresponding to each attack scenario, which will be discussed subsequently.

As noted earlier, a number of requirements that the ES&S system must satisfy are enumerated in the ES&S system manual (Inc, 2007) (such as configuration instructions and the user's manual) and a corresponding video (Election and Systems, 2009), which describes how the system works on election day. Instead of describing properties such as in Sastry (2007) – e.g., "A ballot cannot be cast without the voter's consent to cast it; the DRE only stores ballots that have been confirmed by the voter.", or in McGaley (2008) – e.g., "The e-voting system shall be protected against threats to its availability including: malfunction, breakdown and denial of service attacks." – we rearrange and split the properties so that providing their equivalent formal specification is fairly manageable, if not easy. The link from the concrete requirements listed below to abstract or generic voting machine requirements (e.g., availability, accuracy, privacy, fairness, eligibility) are given elsewhere, such as in Kremer and Ryan (2005) and Delaune et al. (2009). However, and if necessary, we highlight (within a bracket) such abstract requirements with properties we list below.

In the following, we present a sample list of the most important critical security requirements that the ES&S voting system must meet. The list is by no means exhaustive, but it is chosen to reflect important properties that are essential building blocks for most DRE e-voting machines equipped with an RTAL/VVPAT. Notice that the presentation of the requirements does not follow any order of importance, nor is it in sequence.

A correctly functioning DRE must satisfy the following properties.

Property 1. The same CF Card must be present throughout the voting session (availability).

Property 2. The DRE must authenticate the PEB using the EQC, and the same master PEB must be used to open and close the terminal (eligibility).

Property 3. Display screens presented to the voter must accurately reflect the ballot downloaded from the PEB and the selections made by the voters.

Property 4. The DRE terminal only allows two valid actions for the voter until he/she reaches the final review (vote summary) screen: (1) select or cancel a candidate on the screen or (2) move forward or backward through the ballot.

Property 5. For each valid voter action (i.e., starting to vote, making a select or cancel, and finishing a vote) the DRE must enable the RTAL to record the action on the RTAL tape accordingly.

Property 6. The DRE must automatically forbid an overvote.

Property 7. The DRE must report undervoted races, if they exist, and the review screen must display the message "BALLOT NOT COMPLETED".

Property 8. When the voter confirms his/her ballot, the ballot images recorded in the local storage must correctly reflect the selections made by the voter (voter verifiability and cast-as-intended).

In other words, Property 8 states the fact that the DRE must not change the ballot after the voter chooses their candidates.

Property 9. The DRE terminal should start chirping if there is no input from the voter for 10 time units since the last input but not after he/she confirmed.

The RTAL must satisfy the following properties:

Property 10. The RTAL should scroll up a minimum distance after the summary has printed, in order to move the previous vote out of sight (anonymity).

Property 11. The RTAL must update the paper tape after the voter pushes the start button, makes a choice (select or cancel), confirms a vote, or when the poll worker rejects the ballot of a fleeing voter.

Even if Property 3 and Property 11 state different requirements, they are meant to express the fact that the voter must have a chance to preview (both on the DRE screen and on the RTAL window) the contents of the ballot and accept or reject it.

The PEB must satisfy the following properties:

Property 12. The election-specific secret code (EQC), which is a 32-bit (4 digit) code, must be present on a PEB and must always match with the one stored inside the DRE; otherwise, the PEB should be rejected by the DRE terminal whenever the poll worker attempts to insert it.

Property 13. At the end of the election, the copy of the ballot images downloaded from the DRE must be the same as the ballot images that were loaded into the DRE prior to starting the election.

The CF Card should satisfy the following property:

Property 14. The poll closing procedures must copy the audit information (such as the event log) accumulated in the local storage to the CF Card.

The following global properties must be ensured by the system components all together:

Property 15. No discrepancy should be observed among the following: (1) the individual cast ballot records (or ballot images) recorded by the machines; (2) the summary tape generated on Election Day at the close of polls on individual machines; (3) the totals that were accumulated and reported by the DRE and RTAL (counted-as-cast).

The above requirement can further be refined into the following requirements.

Property 16.1. The vote entries printed on the RTAL tape during and after the election must be equal to the ballot records cast plus the rejected votes in the DRE.

Property 16.2. The number of fleeing voters recorded in the audit log file, which is downloaded into the CF Card, must be equal to the number of rejected ballots printed on the RTAL tape.

Property 16.3. The undervoted races in the audit log file, which is downloaded into the CF Card, must be equal to the undervoted races that have been reported on the RTAL tape.

Property 16.4. After the voting is closed, the results downloaded into the master PEB must be equal to the sum of the results collected from each DRE; furthermore, it must be equal to the sum of the printed paper tapes from all RTALs.

The above requirements are converted into ASTRAL specifications in Section 5.2. It is also worth remarking that we do not specify nor analyze usability requirements, such as "the voting device must not display any information about the voter's selections outside the vote casting interface; the vote casting interface must clearly indicate to the voter whether the voting device is in an active state or an inactive state." However, we consider specifying the possible states of the machine (e.g., the machine is in *poll worker*, *voter*, *sleep* or *chirping* mode), since such information helps us understand the different operations that a poll worker or a voter experiences when interacting with the machine.

3.4. Selected attack scenarios

Like any other voting system, the ES&S voting system can be subjected to attack by a number of different types of attackers with different capabilities. An attacker can be an *outsider* (have no special access to any of the voting equipment), a *voter* (have limited and partially supervised access to voting systems during the process of casting their votes), *a poll worker* (have extensive access to polling place equipment), *an election official* (have extensive access both to the back-end election management systems and voting equipment), and more.

Next, we give a short overview of selected attack scenarios that are discussed for the ES&S system in the EVEREST report. We assume that voters can leave the voting booth without checking the votes shown on the confirmation screen – i.e., leaving the voting booth without completing the voting – as observed in practice. These types of voters are called fleeing voters. We now present four selected attack scenarios for which we give formal specifications later. For the sake of understanding, we also sketched sequence diagrams corresponding to each attack scenario. Here we emphasize that our aim is not to provide a formal semantics for the attacks using UML sequence diagrams (or another modeling language) nor to challenge the expressiveness of UML notations. The attack scenarios are all alterations of the normal voting process, which is shown in Fig. 2.

(i) *Changing the vote for an unattentive voter.* In this scenario, the voter proceeds with the normal voting process (as in Fig. 2) and the attacker intercepts the process just before the review ballot is displayed. The attacker steals votes by assigning them to the candidate who s/he desires to win. The modified vote is displayed on the DRE review screen and the change is printed on the RTAL tape. If the voter does check the screen or the printed output and discovers that an error has been made, s/he can recast the vote and the attacker will stop stealing votes for a period of time. However, if the voter is unattentive, the attacker's modification will be stored locally upon the voter's confirmation (see Fig. 3). This attack is more interesting (and meaningful) if the voter is able to change their vote (using

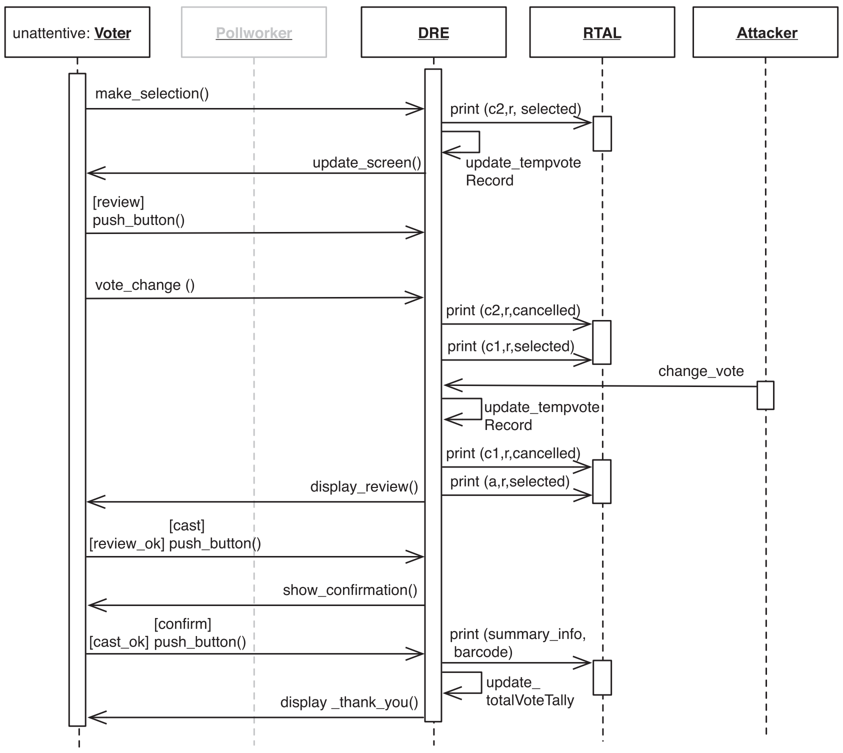


Fig. 3. Changing an unattentive voter's vote.

vote_change) and the DRE is unable to tell where vote change requests are coming from.

(ii) *Changing the vote for a careful voter.* This scenario assumes the voters carefully cast, check the screen and printout, and confirm. However, they are not familiar with all the details of how their votes are printed on the RTAL tape. The attacker does not intercept the normal voting process until after the cast ballot and confirmation screens have been shown to the voter. At this point, the attacker changes the voter's electronic ballot, and the RTAL prints the modified selection. The RTAL then immediately prints the summary information along with the barcode.

(iii) *Canceling or completing the vote for a fleeing voter.* In this scenario the attacker takes advantage of a fleeing voter, a voter who does not complete the voting procedure, by intercepting the call to the routine that enables a chirping sound. In the ES&S machine, this chirping sound alerts the poll worker that a voter has fled. There are two possible scenarios depending on the voter's vote:

1. If the fleeing voter voted against the attacker's candidate, then the attacker does nothing and lets the chirping routine perform as it should (see Fig. 4(a)). The poll worker then discards[1] the ballot and there will be one less vote for the undesired candidate.
2. If the fleeing voter voted for the attacker's candidate but s/he did not complete the voting process then the attacker completes the voting process (see Fig. 4(b)). This results in another vote being cast for the attacker's candidate.

(iv) *Faking a fleeing voter to cancel a vote.* This attack scenario is similar to the third attack scenario. However, in this case the attacker cancels the vote by making it look like the voter fled. In particular, if the voter did not choose the candidate that the attacker wants, the attacker intercepts the confirmation process and pretends to cast the ballot: the normal "thank you" screen is displayed, but nothing is printed on the RTAL tape. After some amount of time elapses (during which the voter most likely leaves the voting booth) the attacker directs the system to display the confirmation screen. Then after another reasonable amount of time has passed the attacker calls the chirping sound routine and the machine immediately starts chirping. A poll worker will think the voter was a fleeing voter and the ballot will be discarded (see Fig. 5).

In the above attack scenarios, various low level details that are not the interest of formal specification and verification are omitted. Moreover, the four attacks given above are by no means exhaustive and they do not represent all the different types of attacks discussed in the EVEREST report for the ES&S system. Instead, they are attacks that we believe demonstrate the flavor of this work. It is also important to clarify that any formalization must be at a given level of abstraction. The kinds of properties that can be expressed and proved depend upon such level. Our formalization is no exception and can not model all kinds of attacks that can be performed on the machine or the low level details of some attacks. The goal, however, is not that of being all-encompassing, but rather, that of complementing existing technologies and filling an existing gap in the formal verification of e-voting systems.

4. Overview of the ASTRAL language

ASTRAL (Kolano et al., 1999) is a high-level formal specification language designed for reactive systems. The language constructs allow one to build modularized specifications of complex systems using state machines. ASTRAL provides a mechanism for specifying critical system requirements as first order formulas, and a formal proof system for proving that the system actually meets the stated requirements. The language is intended to be executable, which in turn allows developers to treat specifications as prototypes.

[1] In Ohio the votes of fleeing voters are discarded. In contrast, in California the poll worker casts these votes.

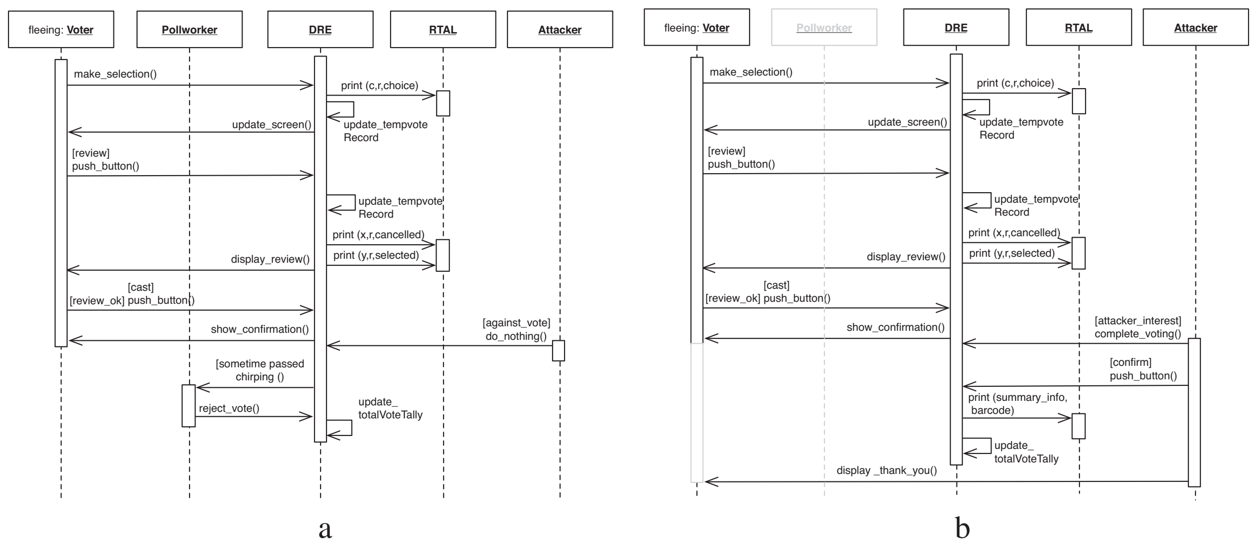


Fig. 4. Canceling or completing the vote for a fleeing voter. (a) Canceling the vote. (b) Completing the vote for a fleeing voter.

An ASTRAL specification of a system consists of a global specification and process specifications. The global specification contains declarations of the process instances, global constants and non-primitive types (which may be shared by process instances), and system level critical requirements. An ASTRAL process specification presents an abstract model of what constitutes the process (types, constants, variables), what the process does (state transitions), and the critical requirements the process must meet. The process being specified is thought of as being in various states, with one state differentiated from another by the values of the state variables, which can be changed only by means of state transitions. A transition is modeled by entry and exit conditions, and a non-zero duration is assigned to each entry/exit pair. Specification exceptions are handled explicitly by adding except/exit pairs in addition to the normal entry/exit pairs. Transitions are executed as soon as the entry conditions are satisfied assuming no other transition for that process instance is executing.

Every ASTRAL process can export both state variables and transitions. As a consequence, the former are readable by other processes while the latter are executable from the external environment. Interprocess communication is accomplished by broadcasting the value of exported variables, as well as the start and end times of exported transitions. In addition to specifying system state (through process variables and constants) and system evolution (through transitions), an ASTRAL specification also defines system critical requirements and assumptions on the behavior of the environment that interacts with the system. The behavior of the environment is expressed by means of environment clauses, which describe assumptions about the pattern of invocation of external transitions. Critical requirements are expressed by means of invariants, constraints and schedules. The invariants express the critical requirements that are to hold in every reachable state. That is, they state properties that must initially be true and must be guaranteed to hold during system evolution. The constraints express the

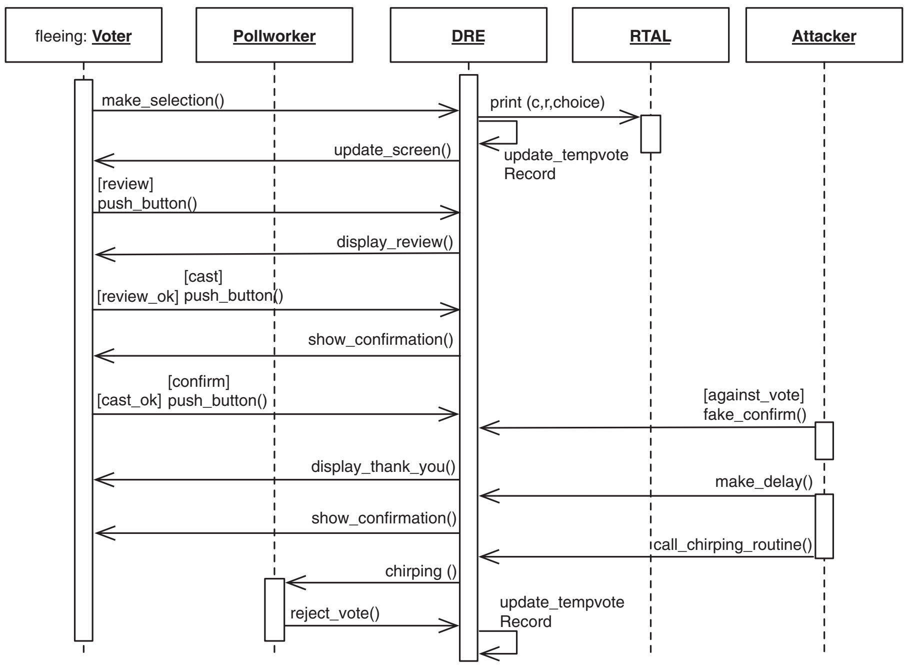


Fig. 5. Canceling a vote by faking a fleeing voter.

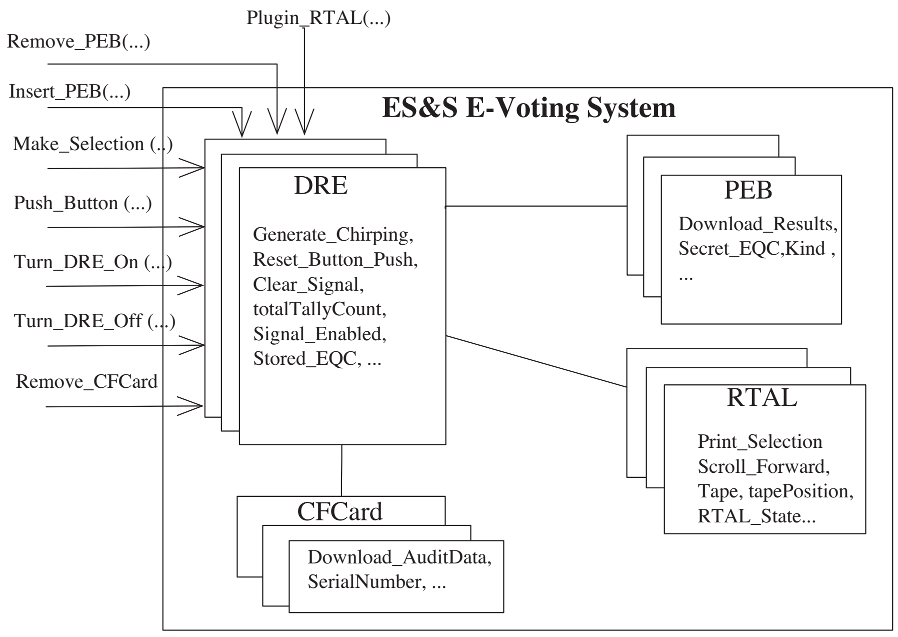

Plugin_RTAL(...)
Remove_PEB(...)
Insert_PEB(...)
Make_Selection (...)
Push_Button (...)
Turn_DRE_On (...)
Turn_DRE_Off (...)
Remove_CFCard
ES&S E-Voting System
DRE
Generate_Chirping, Reset_Button_Push, Clear_Signal, totalTallyCount, Signal_Enabled, Stored_EQC, ...
PEB
Download_Results, Secret_EQC,Kind , ...
RTAL
Print_Selection Scroll_Forward, Tape, tapePosition, RTAL_State...
CFCard
Download_AuditData, SerialNumber, ...

Fig. 6. A simplified view of an ES&S voting system. All the interactions are typically done through the DRE.

critical requirements that must hold between any two states that correspond to the start and end of a transition. Note, however, that the requirements contained in a constraint could be expressed in an invariant, and thus the constraint is just a notational convenience (Kolano et al., 1999). Invariants can be global or local; the global invariants represent properties that need to hold for the realtime system as a whole, while local invariants and constraints defined at the process type level represent properties that must hold for each process instance. Invariant and constraint properties must be true regardless of the environment or the context in which the process or system is running.

Our choice of ASTRAL is twofold. First of all, ASTRAL is a more expressive language for real-time systems, and (as noted earlier) we treat the ES&S voting system as a complex, real-time embedded system. Therefore, the language suits our purpose. The second motivation is related to the nature of the ES&S voting system (generally, true for other voting systems), which consists of several variables representing the different behaviors of the voting processes and its requirements. In fact, different modeling languages are more or less suited to the verification of different critical requirements. In theory, explicit state model checking is a rigorous method. Unfortunately, model checking can only provide this rigor for reasonably small specification, since the number of states rapidly exceeds computational limits for complex specifications (like in our case) and is unfeasible for the model analysis. An alternative approach to verification without model checking is theorem proving. This allowed us to experiment and compare different approaches (theorem proving and model checking) to the verification of e-voting systems. See Villafiorita et al. (2009) and Tiella et al. (2006) for a discussion of the verification using model checkers.

5. Formal analysis of an e-voting system

We now present the specification and verification of the ES&S voting machine by showing a sampling of the specification that, we believe, provides the flavor of the work. On top of the assumptions we mentioned previously, to make the specification and analysis simple but without losing generality, one DRE machine per polling station is assumed. We assume also that there is one CF Card, one RTAL, and one PEB (master) per DRE machine used in the election; moreover, we assume that there is one race per screen.

5.1. *ASTRAL specification of the ES&S system*

We formulate each component of the ES&S voting system as an ASTRAL process instance (see Fig. 6).

There is a process specification for each process type declared in the global specification – i.e., four process types are declared in the global specification of the ASTRAL model of the ES&S system. Below is an example of a process declaration:

```
PROCESSES
  the_DRE: array [1..Number_Of_DRE] of DRE_Process,
  the_RTAL: array [1..Number_Of_RTAL] of RTAL_Process,
```

We declared user defined types and constants to represent useful concerns about the ES&S system inputs and outputs, like in the following snippet specification:

```
TYPE
  DRE_ID: TYPEDEF p: ID (IDTYPE(p) = DRE_Process),
  PrintValue,/* unspecified type*/
  Title IS SUBTYPE OF String,
  Candidate_Name IS SUBTYPE of String,
  DecisionType: (Selected, Canceled),
  Button: (RESET, EXIT, CANCEL, CLOSE,_START, NEXT,
     BACK, REVIEW, CAST, CONFIRM),
[...]
CONSTANT
  Installed_CFCard(DRE_ID): CFCard_ID,
  Plugged_In_RTAL(DRE_ID): RTAL_ID,
  Make_Print_VoteEntry (Name, Title, Decision): PrintValue
```

The `DRE_ID` line declares `DRE_IDs` to be exactly those ids that are process instances of type `DRE_Process`. The `DecisionType` and `Button` are enumerations, that represent, respectively, the voter's decision on a candidate for a given contest and the buttons that can be used to interact with the touchscreen. In contrast, the first two constants associate each DRE with a unique CF Card and RTAL printer, which take `DRE_ID` as an argument and return `CFCard_ID` and `RTAL_ID`, respectively. `Make_Print_VoteEntry` represents the print format on the RTAL paper tape when the voter selects or cancels a particular candidate. In addition to printing vote selection, the RTAL also prints start and summary information for each voting session.

5.1.1. *Modeling the DRE process*

The ES&S DRE device is modeled by the process type `DRE_Process`. The initial clause of the DRE model states that a CF Card is inserted in the machine and that a unique RTAL printer is attached to the DRE.

```
INITIAL
  EXISTS f: CFCard_ID
    (f = Installed_CFCard (Self)
      − > Which_CFCard_Installed = f
      & CFCard_Installed = TRUE
      & CFCardSerialNumber =
        Which_CFCard_Installed.SerialNumber)
    & EXISTS rt: RTAL_ID
      (rt = Plugged_In_RTAL (Self)
      − > Which_RTAL_Plugged_In = rt
        & RTAL_Plugged_In)
```

Using the import clause in the interface section of `DRE_Process`, the process can import globally declared types, constants, and definitions, as well as variables and transitions exported by other processes in the system. For instance, `Installed_CFCard` and `Plugged_In_RTAL` are constants declared in the global specification and are imported using the import clause of the `DRE_Process` process. Similarly, the process can also export variables and transitions which can be used by other processes. For instance, `Which_CFCard_Installed` below is an exported variable:

```
VARIABLE
  NumberOfSelected (Race_Num): Non_Negative,
  totalTallyCount(Candidate_Name,Title):Non_Negative,
  Which_CFCard_Installed: CFCard_ID
```

The DRE machine stores vote records locally and automatically forbids overvotes, but not undervotes. The number of candidates currently selected for a particular race and the total number of votes for a particular candidate in a race are modeled with the first two variables above.

The communication between the DRE and the RTAL processes is modeled by the exported variables:

```
VARIABLE
  Signal_Enabled: Boolean,
  Which_Signal: SignalType
```

where the first variable signals that the DRE is sending information to the RTAL printer and `Which_Signal` carries the kind of information to be printed (e.g., is the print information, a start vote session message or a vote selection).

To model permissible operations on the DRE machine, it is also necessary to capture the phases of the election and the various modes of the terminal during election day. We use the following variables:

```
VARIABLE
  Which_Phase: Voting_Phase,
  Terminal_Mode: Mode,
  DRE_State: Terminal_State,
```

These variables indicate, respectively, that the phase of the election (pre-voting, during voting, and post-voting phases), the terminal mode, and the state of the poll (opening, opened, closing, or closed). The last two variables are only meaningful during the actual election day – i.e., `Which_Phase = During_Voting`.

When a voter casts a vote, s/he is actually interacting with the system by navigating from one screen to another using an appropriate button (such as `NEXT` or `BACK`). We model such interaction by assigning an integer number to each screen shown to the voter and by defining a function that takes as input a screen number and returns the information to be displayed and the buttons available. The variable `Display` of type screen, is used to hold the state of the screen as it is to be shown to the voter while s/he is voting. For example, if the voter is in one of the race screens then the value of the `Display` contains the candidates of that race with appropriate button(s) displayed on it.

Once we capture the relevant data structures that allow one to hold information about the DRE, the next step is modeling the behavior of the DRE itself. This is modeled by ASTRAL transitions. Twelve transitions are used to model the possible operations of the DRE machine. For instance, the `Insert_PEB` exported transition models the insertion of a qualified PEB device in order to allow various operations to run the election; the `Initialize_Ballot` transition models the initialization of a ballot when a qualified voter comes, and, the `Push_Button` exported transition specifies the behavior of the DRE while the voter and/or poll worker interacts with the system by navigating from one screen to another using an appropriate button.

An ES&S DRE requires a poll worker to insert a qualified PEB device in order to allow various operations to run the election. These operations include loading the appropriate ballot, opening or closing polls, initializing the ballot, collecting election results, and performing various administrative tasks. We modeled all these aspects with appropriate transitions.

The following snippet specification encodes the ballot loading operation prior to start election.

```
TRANSITION Insert_PEB (p: PEB_ID)
ENTRY [TIME: I_P_Dur1]
  MachineTurnedOn & Stored_EQC = p.Secret_EQC
  & p.Kind = Master & ~PEB_Inserted
  & Terminal_Mode = Deactivated & Which_Phase = Pre_Voting
  & DRE_State = Initial_State & ~Ballot_Loaded
  & FORALL R: Race (Race_Candidates (R) = EMPTY)
EXIT
  Which_PEB_Inserted = p
  & PEB_Inserted
  & FORALL R: Race (
    Race_Candidates(R) = P.Candidates_Of_Race(R))
  & Ballot_Loaded
[...]
```

Once the ballot is loaded while inserting the PEB, the poll worker must remove the inserted PEB safely. This is done by calling `Remove_PEB` transition. The result of the remove operation, in the nominal case, is changing the state of the system to allow voting and putting the machine in sleeping mode (i.e., `Which_Phase = during_voting`, `DRE_State = Opened`, and `Terminal_Mode = sleep_mode`). In other words, this indicates that it is now voting time, the poll is opened for election, and the terminal mode goes to sleep.

The initialization of a ballot when a qualified voter comes is specified in the model by the transition

```
TRANSITION Initialize_Ballot
  ENTRY [TIME: I_B_Dur]
    DRE_State = Opened
    & Terminal_Mode = pollworker
    & EXISTS p: PEB_ID
      (Which_PEB_Inserted = p
        & PEB_Inserted))
  & Proceed_Ballot_Init & ~Ballot_Initialized
EXIT
  FORALL R: Race (
    Displayed_Candidates (R) = { SETDEF
    C: Candidate (C ISIN Race_Candidates (R))})
  & FORALL R: Race, C: Candidate (
    C ISIN Displayed_Candidates (R)
    & ~Picked (Candidate_Name(C),Race_Title(R)))
  & FORALL R: Race (Number_Of_Selected(R) = 0)
  & Ballot_Initialized & ~Proceed_Ballot_Init
  & underVotedRaces = EMPTY
```

The entry conditions specify that the poll has to be opened in the poll worker terminal mode, the PEB is inserted, and the ballot has not been initialized for the voter who is ready to cast her/his vote. It should be noted that the voting procedure usually allows voting after scheduled poll closing time as long as a qualified voter is still in line. The exit condition specifies that all the variable values from the last voter are reset – i.e., the `Picked` value for each candidate–race pair, the number of selections for each race, and the temporary vote list are all reset. Therefore, the ballot is ready for the next voter, and the local variable `Ballot_Initialized` is set to true.

There are four nominal situations in which the PEB can be removed:

1. after the poll worker loaded ballots prior to opening the DRE terminal for voting;
2. after the poll worker initialized the ballot for the next voter during the voting phase;
3. after the poll worker performed administrative operations (such as after correcting the chirping terminal mode) during the voting phase;
4. after the poll worker downloaded the election results after the terminal is closed for election.

If the `Remove_PEB` transition has been fired because the poll worker initialized the ballot, then the terminal mode changes to voter mode, the current screen becomes the starting screen for the eligible voter with a `START` button on it:

```
[...]
/*Removing the PBE after the ballot has been initialized for the voter.*/
Terminal_Mode = voter_mode
& scrName = START_SCREEN
& scrNumber = 0
& Screen_Buttons(scrNumber) BECOMES {START_BUTTON}
& Min_Display (scrNumber) BECOMES
Display_Info (Push_Start_Button_To_Start_Voting
    ,Screen_Buttons (scrNumber))
  [...]
```

In a touch-screen based voting system, a voter makes a choice or changes a previous choice by touching the candidate name on the display. In either case, the DRE must capture and process the touch correctly. `Make_Selection` is an exported transition, which must be called by the voter.

```
TRANSITION Make_Selection (cName: Name)
  ENTRY [TIME: M_S_Dur]
    Which_Phase = during_voting
    & Terminal_Mode = voter_mode
    & Race_Screen (scrNumber)
    & currentRace = Which_Race(scrNumber)
    & Display(scrNumber) =
      Display_Contest (Race_Title (currentRace)
      ,Displayed_Candidates(currentRace)
    ,Screen_Buttons (scrNumber))
    & EXISTS C: Candidate (
      C ISIN Displayed_Candidates (currentRace)
      & Candidate_Name (C) = cName)
  & ~Signal_Enabled
```

It specifies the occurrence of a screen touch on a particular candidate's name. On entry, the DRE checks that the voter is voting during voting period, the terminal is in voter mode, the current screen is a race screen displaying both the current race with its candidates and the button(s) required to navigate through the screen, the touched candidate `cName` belongs to the displayed candidates, and that the DRE is not currently sending a signal to the RTAL. We used the `Picked` variable to determine whether the candidate has been previously selected. This variable will eventually be used to update the `totalTallyCount` for the selected candidate name `cName` when the ballot is confirmed. The exit assertion for the `Make_Selection` transition is

```
EXIT
/*If the number of selects is greater than the maximum allowed,
the DRE machine should reject the selection locally.*/
/*The parametrized variable Number_Of_Selected keeps track of the number of
  selects.*/
IF ~Picked' (cName,Race_Title (currentRace'))
THEN
  IF Number_Of_Selected' (currentRace') + 1
    <= Max_Choice_Per_Race (currentRace)
  THEN/*over-vote is not attempted.*/
    Number_Of_Selected (currentRace') BECOMES
    Number_Of_Selected' (currentRace') + 1
    & Picked (cName, Race_Title(currentRace'))
      BECOMES TRUE
    & Display (scrNumber') BECOMES
    Update(Display' (scrNumber'),cName,Marked)
    & tempVoteRecord (currentRace') BECOMES
    tempVoteRecord' (currentRace') UNION
    {SETDEF C: Candidate(Candidate_Name(C) =cName)}
    /*set variable value for the RTAL to print.*/
    & pickedName = cName & pickedValue = Selected
    & Signal_Enabled & Which_Signal = Vote_Signal
    & currentRace = currentRace'
  ELSE /*else over-vote is attempted.*/
    Min_Display (scrNumber') BECOMES
    Display_Info (OverVote_Prohibited, NoButton)
  FI
ELSE /*else, cancel the previous choice.*/
  Number_Of_Selected (currentRace') BECOMES
    Number_Of_Selected' (currentRace') - 1
    & Picked (cName,Race_Title(currentRace'))
      BECOMES FALSE
    & Display (scrNumber') BECOMES
    Update(Display'(scrNumber'),cName,UnMarked)
    & tempVoteRecord (currentRace') BECOMES
    tempVoteRecord' (currentRace') SET_DIFF
  SETDEF C: Candidate (
      Candidate_Name (C) = cName)
[...]
```

There are two possible cases when a voter marks a candidate on the screen:

- *Making a selection*. The following scenario occurs: (i) as long as there is no overvote attempted the number of selections for this candidate for the current race is incremented by one, `Picked` is set to true, the current screen is updated, and `cName` is included in `tempVoteRecord`, which will be used to display the voter's final selection when the voter requests a preview. In addition, the exported variables `pickedName`, `currentRace`, `pickedValue` and `Which_Signal` receive new values, and the signaling variable is set to true. This indicates that the RTAL can now print the selection expressed in these exported variables. (ii) Otherwise, the voter attempted to overvote and the DRE will display the appropriate message on the screen.
- *Canceling a previous selection*. In this case, the exit assertion specifies that the number of selected candidates for the current race is decremented by one, `Picked` is reset to false, and `cName` is removed from the `tempVoteRecord`. The rest of the variables are updated accordingly and the cancellation expressed in the exported variables information is sent to the RTAL.

Another important transition to discuss is how we specify `Button_Push`, which is also an exported transition and, therefore, is called by the voter and/or poll worker. Notice that each screen is associated with an integer value and, in some cases, with a particular name. To make the specification simple but without losing generality, we assume one race per screen. In reality, however, a screen can display more than one race. The voter and the poll worker interact with the screen while casting votes and administering the election (such as, loading a ballot prior to starting an election, initializing a voter's ballot, or dealing with abnormal situations). A voter navigates from one screen to another, by calling the transition `Push_Button`. The transition has a number of entry/exit pairs that correspond to the buttons defined previously. The entry and exit assertions that correspond to the `START` button are as follows:

```
ENTRY
  b = START_BUTTON
  & b ISIN Screen_Buttons (scrNumber)
  & scrName = START_SCREEN
  & scrNumber = 0 & ~Button_Pushed (b)
  & Which_Phase = During_Voting
  & Terminal_Mode = voter_mode & ~Signal_Enabled
```

The first five conjuncts specify conditions about the button and the current screen. They specify that the button that the voter pushed is `START_BUTTON`, the button is in the screen button list for the current screen, the current screen is `START_SCREEN`, the corresponding screen number equals zero, and the start button was not previously pushed. The next two conjuncts deal with election period and the status of the DRE terminal. The election phase must be during-voting and the terminal mode is voter-mode. The last conjunct of the entry assertion is used by the DRE to modulate the signaling information in order to alert the RTAL.

```
EXIT
  Button_Pushed (b) BECOMES TRUE
  & scrNumber = 1
  & currentRace = Which_Race (scrNumber)
  & Screen_Buttons(curScreen) = { NEXT }
  & Display (scrNumber) BECOMES
    Display_Contest(Race_Title(currentRace)
      ,Displayed_Candidates' (currentRace)
      ,Screen_Buttons (scrNumber))
  & voterNumber =voterNumber' + 1
  /* Make available for RTAL to print. */
  & Signal_Enabled & Which_Signal = Start_Signal
  & RTALMessage = VOTE_SESSION_STARTED
  /*Once the voter starts voting, Ballot_Initialized is set to FALSE.*/
  & ~Ballot_Initialized
```

The exit assertion for the start case indicates that the voter has pushed the `START` button, the screen number is incremented by one, the current race is updated, the current screen displays the first race, and the only button available to push is `NEXT`, and the number of voters who visited the poll is incremented by one. In addition, the DRE updates the value of the signaling variables and `RTALMessage` to be printed out on the paper tape.

The entry assertions for the rest of the entry/exit pairs are more or less identical to the first five conjuncts of the start case except the button being pushed is different in each case (with additional conjuncts if applicable). The exit assertion, however, for each button push can be different depending on which button was pushed. Below, we discuss the exit assertion for the `CONFIRM` button push.

After the voter has completed all of his/her votes, the voter has to cast and confirm the choices. Once the voter touches the `CAST` button and confirms the vote by touching the `CONFIRM` button, the DRE updates the total tally in the exit assertion of confirm. Note that when the voter reaches the end of the ballot, they will be prompted to press the `REVIEW` button. When the `REVIEW` button is pressed the voter will be notified of any unvoted, or undervoted contests or if the ballot has been left blank. The voter has the option of reviewing their ballot and making any changes (by using the `BACK` button or by touching on the candidate name) before casting their ballot. In this paper we specified the change only by using the `BACK` button push until the voter reaches the screen that contains the candidate. Pressing the `CONFIRM` button will cast the ballot.

```
/*exit assertion for 'confirm' button*/
EXIT
  Button_Pushed (b) BECOMES TRUE
  /*Store the vote locally because the voter has confirmed.*/
  & FORALL C: Candidate, R: Race
    (C ISIN Displayed_Candidates' (R)
    & IF Picked' (Candidate_Name(C),Race_Title(R))
    THEN
    TotalTallyCount (C,R)=TotalTallyCount'(C,R)+1
  ELSE
    NOCHANGE (TotalTallyCount (C, R))
  FI)
& (IF Min_Display' (scrNumber') =
  Display_Info (Ballot_Not_Completed
      ,Screen_Buttons'(scrNumber'))
  THEN
    NumberOfLogEntry = NumberOfLogEntry' + 1
    & EventLog (NumberOfLogEntry)
        BECOMES underVotedRaces'
    & underVotedRaces = underVotedRaces'
    & RTALMessage = BALLOT_ACCEPTED_UNDERVOTE
  ELSE
    RTALMessage = BALLOT_ACCEPTED
    & underVotedRaces = NoUnderVotedRace
  FI)
  & Signal_Enabled & Which_Signal = Summary_Signal
  & BallotBarcode = BARCODE (voterNumber')
  & Terminal_Mode = sleep_mode
  & scrNumber = - 1 & scrName = SETUP_SCREEN
/*Reset the temporary vote record. */
  & FORALL R: Race (tempVoteRecord (R) = EMPTY)
```

In addition to updating the total tally, the DRE keeps track of log data (such as undervoted races, if they exist, in `underVotedRaces`).

In the ES&S voting system, the DRE is also responsible for generating a chirping sound (when a voter flees, i.e., s/he leaves without confirming the vote), clearing the previous signal value and the button push, by performing the transitions `Generate_Chirping`, `Clear_Signal` and `Clear_Button_Push`, respectively. We omit the discussion of these remaining transitions since they follow a similar pattern with the transitions discussed so far.

5.1.2. *Modeling the RTAL process*

The RTAL collects the output sent by the DRE, mostly for auditing purposes. Namely, it prints vote actions exported by the DRE on a paper tape. In our specification, the paper tape contains a list of voter records, where each individual voter record is a continuous sequence of voter actions.

These are captured by the following variables.

```
VARIABLE
  Tape (Pos_Integer): PrintValue,
  tapePosition: Tape_Number,/*positive integer*/
  RTAL_State: RTALState,
  summaryPrinted: Boolean,
  VoteStartPosition (Voter_Number): Tape_Number,
  VoteEndPosition (Voter_Number): Tape_Number
```

The `Tape` variable represents the RTAL paper tape where the start information, vote selection, and summary information are continuously printed for each voter. After each print, the RTAL `tapePosition` is incremented appropriately. The variables `RTAL_State` and `summaryPrinted`, respectively, are used for keeping track of the current state of the RTAL and determining whether the summary information has been printed. This is to know when to scroll the tape forward by some amount in order to protect the secrecy of the previous voted ballot. Moreover, the variables `VoteStartPosition` and `VoteEndPosition` delineate the voter record on the paper tape. In fact, the model of the RTAL process is similar to an array of continuous values of votes. Each time a voter makes a choice the corresponding record is inserted into the array, and at the end of each vote confirmation empty values are appended to represent the advancing operation of the RTAL.

There are two main behaviors of RTAL that are of interest for the specification: printing and advancing the printer trail after printing the summary information on the tape to keep the vote secret. The former is modeled by the `Print_Selection` transition, and the latter is modeled by the `Scroll_Forward` transition.

```
TRANSITION Print_Selection
  ENTRY [TIME: P_S_Dur]
    My_DRE.Plugged_In
    & My_DRE.Signal_Enabled
    & RTAL_State = Wait
    & My_DRE.Which_Signal ~= NoSignal
```

The first three conjuncts in the entry assertion specify that the RTAL has been plugged into the DRE, that the DRE has sent a signal, and the RTAL is waiting for the DRE signal to print. The fourth conjunct specifies what type of information the RTAL is signaling to print.

```
(IF (My_DRE.Which_Signal = Start_Signal
 |My_DRE.Which_Signal = Vote_Signal)
  THEN
  tapePosition = tapePosition' + 1
  & CutLengthCounter = CutLengthCounter' + 1
  & (IF My_DRE.Which_Signal = Start_Signal
    THEN
    [...]
  ELSE/*voting entry printing*/
    Tape (tapePosition) BECOMES
    Make_Print_VoteEntry (My_DRE.pickedName,
    My_DRE.currentRace, My_DRE.pickedValue)
  FI)
    ELSE/*Summary printing*/
    tapePosition = tapePosition' + 3
    & CutLengthCounter = CutLengthCounter' + 3
    & Tape (tapePosition - 2) =
    Make_Print_Info (My_DRE.RTALMessage)
    & Tape (tapePosition - 1) =
    Make_Print_Undervote (My_DRE.underVotedRaces)
    & Tape (tapePosition) =
    Make_Print_BallotBarcode (My_DRE.BallotBarcode)
    & (FORALL i: Tape_Number
      (i ~= tapePosition & i ~= tapePosition - 1
      & i ~= tapePosition - 2
      -> NOCHANGE (Tape (i))))
    & VoteStartPosition (voterNumber) BECOMES
    tapePosition - CutLengthCounter + 1
    & VoteEndPosition (voterNumber) BECOMES tapePosition
  & summaryPrinted
FI)
```

After the transition is fired, depending on what signaling mode has been received by the RTAL, the corresponding entry is printed. Notice that each vote record is uniquely identified by a `barcode`, which encodes the voter's ballot selections in the RTAL record without revealing the identity of the voter. This `barcode` is printed on the tape along with the summary information of the vote entry. Upon the completion of printing the summary information, the printer is also scrolled forward by calling `Scroll_Forward` transition.

5.1.3. *Modeling the PEB and CF Card processes*

The PEB device is specified by an instance of type `PEB_Process`. As mentioned earlier, the PEB device – in addition to being used to load the ballot data into the iVotronic terminals prior to starting the election and to initialize a ballot when a voter comes during election – is used to transfer election specific data between Election Central and poll locations. This data is represented by the variables `Candidates_Of_Race`, `tabulatedData`, and `copyOfBallotImages`.

We mentioned that, according to the ES&S voting process specification, the DRE authenticates each PEB by its four digit EQC code (represented by the `Secret_EQC` variable). While all PEBs are internally identical in construction, they are discernible from one another by the read-only information burned in the PIC: their serial number, and more importantly by their PEB kind, namely either "master" or "supervisor". In our specification, we only use PEB kinds to distinguish PEBs (i.e., `Kind: PEBKind`).

The most important aspect to specify about the behavior of the PEB process is that after the terminal is closed, the poll worker uses the master PEB to collect and store the tabulated data and copies of the "images" of the ballots. This is specified by the transition `Download_Results`.

```
TRANSITION Download_Results (D: DRE_ID)
[...]
/*Download the election result.*/
FORALL C: Candidate, R: Race
  (C ISIN D.Race_Candidates (R)- >
    tabulatedData(C, R, D) = D.TotalTallyCount(C, R))
    /*Dump copy of ballot images into this PEB.*/
  &copyOfBallotImages (D) BECOMES
  Download_BallotImage (
    {SETDEF Pair: Race_Candidates_Pair (
      EXISTS R: Race (Pair [Contest] = R
        & Pair [Nominees] = D.Race_Candidates(R)))}
)[...]
```

The CF Card is specified by an instance of type `CFCard_Process`. An audit file is automatically saved to the card by calling the `Download_AuditData` transition (not shown in this paper) when the polls are closed. From a formal specification point of view, however, we are only interested in the audit log file, which contains the undervoted races and the number of fleeing voters, indicated by the following variables:

```
VARIABLE
  EventLog (Pos_Integer): Races,
  numOfFleeingVoters: Non_Negative,
  visitedNumberOfVoters: Non_Negative,
  ADDownload_Completed: Boolean
```

We mentioned that there is one CF Card per DRE machine used in the election. This card is uniquely identified by its serial number – i.e., `SerialNumber: Digit_List`. When the polls are closed, an audit file is automatically saved to the CF Card. In other words, upon closing the terminal while the master PEB is inserted, the DRE automatically enables the CF Card to save audit data. Downloading the audit log file is modeled by the `Download_AuditData` transition.

In summary, the complete ASTRAL specification of the ES&S voting process is approximately 1800 LOC (about 33 pages long). We skipped a number of descriptions about the specification of each component.

5.2. *Critical requirements specification*

After we specify the relevant information of the system model we need to specify what security requirements the system should meet given the assumptions about the behavior of the system and the external environment that interacts with the system.

In particular, we specified the following concerns:

1. *Environmental/procedural assumptions.* We have specified a number of behaviors about the external environment that the e-voting system relies on. For instance, the behavior of the people (voters, poll workers, and election officials) who interact with the system. These are requirements on the environment to ensure fair elections. This is outside the ES&S system, but it influences how the system operates.
2. *Security requirements.* We have specified 25 critical requirements (expressed as invariants, constraints, and schedules) that must be satisfied by the system given all the possible assumptions about the environment. In the ES&S system, for instance, the DRE should correctly handle vote selection and the RTAL should update the paper tape after the voter pushes the start button, makes a selection, confirms a vote, or when a poll worker rejects the ballot of a fleeing voter.

With respect to the first, the DRE, in fact, cannot control the behavior of the voter when s/he interacts with the screen. For example, if the voter touches the candidate name faster then the DRE can process the touches, the normal functioning of the e-voting system may be disrupted. In addition, the procedures that control the voting process are completely outside of the e-voting system, e.g., the poll worker has to wait some amount of time to remove the PEB after loading the ballot, or after activating the ballot for the next active voter. However, they are equally important to carry out a correct and secure election. Therefore, we need to express these concerns in order to guarantee the critical requirements that the system should meet.

```
ENVIRONMENT
/*min_pause is the minimum time between two subsequent selections */
  (FORALL t: time (Call [2] (Make_Selection, t)
    -> Call (Make_Selection) - t > min_pause))
/*min_pause is the minimum time between two subsequent button pushes */
  & (FORALL t: time (Call [2] (f, t)
    -> Call (Push_Button) - t > min_pause))
/*Remove_PEB will be called after the ballot is loaded into the DRE and
  Notify_Time1 units have elapsed. */
& (EXISTS t: Time, p: PEB_ID
  (Now>= t + Notify_Time1
  & p = past (Which_PEB_Inserted, t)
  & past (PEB_Inserted, t)
  & past (Ballot_Loaded, t)
  & past (p.Kind, t) = Master
  & past (DRE_State, t) = Opening
    -> Call(Remove_PEB, t + Notify_Time1)))
```

For instance, the above environment clause for the DRE process states that there must be a minimum pause between two subsequent selections and button pushes. It also specifies the fact that the poll worker should only remove the PEB after the loading operation has passed and `Notify_Time1` has elapsed. All these facts are important to prove the critical requirements, in particular requirements that involve exported transitions. The critical requirements listed in Section 3.3 are essential to maintain the integrity of the election results. In fact, the integrity of the election results depends heavily on the integrity of the software and firmware that runs the central EMS and the hardware used. However, this is largely dependent upon a particular implementation and is not in the scope of this specification. Moreover, audit logs serve a vital purpose, as they can alert an auditor of suspicious or uncommon events that occurred, which could indicate the presence of malicious intent against the system. Because of this, it is critically important that an auditor is completely confident that the information retrieved from the audit logs is complete and accurate. Therefore, the security properties we are interested in mainly concentrate on the integrity of election results.

With respect to the second, we formulated each of the critical properties from Section 3.3 as ASTRAL invariants, constraints, or schedules. We now present examples of critical requirement specifications, mostly related to the integrity of election results.

The fact that a DRE is chirping indicates that at least ten units have passed since the last ballot activity. This is expressed by the following local schedule requirement of the `DRE_Process` (Property 8):

```
(Change (Terminal_Mode, Now)
& Terminal_Mode = chirping->
  Call(Make_Selection) - Call[2](Make_Selection)>=10
  |(Now - Change (scrNumber)>= 10
& EXISTS t: Time (t <= Now
  & t > Change [2] (Terminal_Mode)
  & past (Terminal_Mode, t) = voter_mode))
```

It says the mode of the terminal is set to *chirping* if there is no user input to the DRE within ten time units since the last screen change or the last call to the exported transition Make_Selection by the voter.

Below is the specification of Property 6 – i.e., the fact that the DRE must automatically forbid an overvote.

```
FORALL R: Race (
 Change (Number_Of_Selected (R), Now)
   & Number_Of_Selected (R) ~= Number_Of_Selected' (R)
     -> Number_Of_Selected (R) <= Max_Choice_Per_Race (R))
```

The above constraint must be ensured each time a voter makes a selection by calling the `Make_Selection` transition. More specifically, whenever the `Number_Of_Selected` variable for a race `R` is changed to a non-zero value, the new value must be less than or equal to the `Max_Choice_Per_Race` for that race `R`.

We mentioned that the RTAL must print the corresponding voter action on the tape (Property 11). This requirement must be expressed as a scheduling requirement because the printing activity depends on the signal information sent by the `DRE_Process` through the `Signal_Enabled` variable. The schedule clause for the `RTAL_Process` consists of four conjuncts, each corresponding to a scheduling requirement. Below, we present one of them.

```
(My_DRE.Signal_Enabled
  & past (My_DRE.Which_Signal,
    Change (My_DRE.Signal_Enabled))=Vote_Signal
  & Now-Change (My_DRE.Signal_Enabled)
        > Max_Print_Time
     -> EXISTS t: Time
   (t > Change (My_DRE.Signal_Enabled)
   & t <= Now & Change (tapePosition, t)
   & past (tapePosition, t) =
     past (tapePosition, Now -
       Change (My_DRE.Signal_Enabled)) + 1
   & past (Tape (tapePosition), t) =
   Make_Print_VoteEntry (My_DRE.pickedName,
     My_DRE.currentRace, My_DRE.pickedValue)))
```

This specifies that the vote entry (i.e., a record that consists of a candidate, race, and value of the selection) will be printed on the RTAL tape one tape position below the previous print if the DRE has enabled the signal, made available the information to print, and enough time has elapsed for the choice to be printed; we omit the start and summary conjuncts.

We next consider specifying the integrity of the election results (Property 16.4). This property must guarantee that, after the election is closed, the results downloaded into the master PEB must be equal to the sum of the results collected from each DRE. The property is specified in the global invariant clause as

```
   /*After the election, results downloaded into the master PEB must be equal to the
 results produced by all DREs. */
 EXISTS p: PEB_Number
   (the_PEB [p].Kind = Master
 & FORALL d: DRE_Number(
   the_PEB[p].ResultDownload_Completed (the_DRE[d])
   & the_DRE[d].Which_Phase=Post_Voting
   & the_DRE[d].DRE_State=Closed
   & FORALL C: Candidate, R: Race
     (the_PEB [p].Candidates_Of_Race(R) =
     the_DRE [d].Race_Candidates (R)
   & C ISIN the_DRE [d].Race_Candidates(R)->
   the_PEB[p].tabulatedData (C, R, the_DRE [d])
     = the_DRE [d].TotalTallyCount (C, R))))
 /*Downloaded results in the master PEB must be equal to the printed votes on the
 RTAL tape.*/
 & EXISTS p: PEB_Number (
   the_PEB [p].Kind = Master
   & FORALL d: DRE_Number, rt: RTAL_Number (
   the_RTAL[rt]=Plugged_In_RTAL(the_DRE[d])
   & the_PEB [p].ResultDownload_Completed (the_DRE[d])
   & the_DRE [d].Which_Phase = Post_Voting
   & the_DRE [d].DRE_State = Closed
   & FORALL C: Candidate, R: Race (C ISIN
     the_DRE [d].Race_Candidates (R)->
     the_PEB [p].tabulatedData(C, R, the_DRE [d])
       = CountSelected (C, R, the_RTAL [rt])
         - CountCancelled (C,R, the_RTAL [rt]))))
```

The first conjunct of the invariant specifies that there exists a PEB `p`, such that for every DRE `d` in the precinct, if `p` is the master PEB used in `d'` s terminal to download the election results after the terminal is closed and the election has ended (i.e., `Post_Voting` phase), then the election results for each candidate `C` who ran for race `R` stored in `p` is exactly equal to the total tally counted on `d`'s terminal for candidate `C`. Similarly, the second conjunct specifies that for every RTAL printer `rt` and DRE `d` in the precinct, if `rt` is the printer used by `d` during the voting period, then the election result for each candidate `C` who ran for race `R` stored in `p` is the difference between the total number of selected and the total number of canceled votes printed on `rt`.

In the above specification, the `CountSelected` and `CountCancelled` are definitions that make the specification more readable. More specifically, they respectively introduce predicates which are used in our specification of the voting process to specify how many selections and cancelations have been printed for each candidate `C` who ran for race `R` on the RTAL printer `rt`.

This way, the requirements listed in Section 3.3 are converted into ASTRAL invariants, schedules, and constraints for each corresponding process instance. We need to be clear that we did not convert all the requirements to their ASTRAL equivalent in the way we describe them informally.

6. Extending the system specification by modeling attack scenarios

Analyzing a system in *non-nominal* situations – where some of its components are not behaving in the way they should be – has always provided important insights into the behaviors of a system. This approach is common in system engineering when performing safety assessment of critical systems. When performing safety assessment, the model is augmented with non-nominal behaviors (e.g., due to malfunctioning or attacks). The augmented model (also called the extended model) is then analyzed to understand under what (non-nominal) conditions critical requirements are not met anymore.

We want to take a similar approach to the analysis of the security-critical properties articulated previously. The extended model is a combination of the original specification of the ES&S system that was discussed previously and the attack scenarios given in Section 3.4. More specifically, we extended the original specification with a set of transition specifications that represent known attacks that have been shown to successfully compromise the ES&S system. Each transition corresponds to a particular threat action for the voting system. Thereafter, we process the extended specification and automatically generate proof obligations related to the security requirements for the PVS analysis tool.

In particular, we wish to prove the security properties against the extended model for the following reasons:

- If all of the proof obligations were to be proved, then the system specification must be missing some critical security requirements, since the modeled attacks were already demonstrated to be successful. Therefore, it would be necessary to see what additional critical requirements are needed to disallow the threat actions and keep the extended specification from being proved.
- In contrast, not being able to prove the extended specification would indicate that one, or more, threat action violates at least one critical security requirement. However, since we know that attacks composed of these threat actions have been used to successfully compromise the system, it also indicates that there could be an implementation or specification error or an unsatisfied procedural assumption that results in the actual system or the environment not satisfying their respective formal specification.

6.1. Attack specifications

We model the attack scenarios presented in Section 3.4 in terms of threat actions expressed as ASTRAL transition specifications. The system model is extended by augmenting the specification with new possible states that are the result of the execution of the threat actions.

In particular, the attack scenarios are encoded to extend the original specification of the ES&S voting system using the following strategies:

1. we define new types, variables, and constants. There are two kinds of variables that we declare: those that provide additional information about the state of the system (e.g., the system is now about to display the review ballot) and those that hold information about the successful execution of a threat action (e.g., a fleeing voter has been faked).
2. a transition is defined for each threat action, which is part of a given attack scenario. Note that one attack scenario can be implemented using one or more threat actions.
3. a transition may be split into two or more transitions, or a transition may be extended with more information to specify the attack scenario.

As noted previously, we assume that the attacker can intercept the normal voting process at any point. For instance, if s/he intercepts the process before the review screen is displayed and the attack is successful, then the `tempVoteRecord` variable should include the maliciously modified candidate and the `Display` variable should update the screen accordingly. It is, in fact, the voter's task to correctly verify that what is displayed exactly matches her/his preferences.

To represent the various kinds of voters (unattentive, careful, and fleeing), we introduced the following global type:

```
TYPE
  VoterType: (unattentive, careful, fleeing)
```

In addition, the variables declared, respectively, hold information about whether the voter's vote is changed (obviously, by a successful attack action), whether the fleeing voter is faked, and the name of the attacker's candidate. In addition, information about where the attacker intercepts the process to start the threat action is encoded by the (boolean) variables `review_displayed` and `summary_sent2RTAL`. Namely, they respectively hold information about the attack actions that happened just before reviewing the final votes and right after the summary data is sent to the printer.

```
VARIABLE
  vote_changed, Fleeing_Faked: Boolean,
  attPickedName: Name
```

When an attacker changes or cancels a vote, it is actually performing a sequence of interactions with the DRE process in order to fulfill the threat action. The successful completion of such an action eventually assigns new values to some of the exported variables discussed above.

The following transition specifies the *change vote threat action*, which appears in the sequence diagram depicted in Fig. 3.

```
TRANSITION Attack_Change_Vote
    (vc, ac:Candidate, vType:VoterType)
ENTRY [TIME ACV_Dur]
/*This attack assumes an unattentive voter. This results in a change of vote.*/
  Which_Phase = During_Voting
  & Terminal_Mode = voter_mode
  & vType = Unattentive
  & EXISTS R: Race (
    vc ISIN Displayed_Candidates (R)
    & ac ISIN Displayed_Candidates (R)
    & vc ISIN tempVoteRecord (R)
    & Picked (Candidate_Name (vc), Race_Title(R)))
    & ~Picked (Candidate_Name (ac), Race_Title(R)))
    & vc ~= ac
    & EXISTS b: Button
      (b = REVIEW & Button_Pushed(b))
    & scrName = REVIEW_SCREEN
    & ~Review_Displayed & ~Vote_Changed
  EXIT
    EXISTS R: Race (
    vc ISIN Displayed_Candidates' (R)
    & ac ISIN Displayed_Candidates' (R)
    & vc ISIN tempVoteRecord' (R)
    & tempVoteRecord (R) BECOMES
    (tempVoteRecord' (R) SET_DIFF vc) UNION ac
    & ~Picked (Candidate_Name (vc), Race_Title (R))
    & Picked (Candidate_Name (ac), Race_Title (R))
    & FORALL CN:Name, R1:Title (
      ((CN ~= Candidate_Name (vc)
      & CN ~= Candidate_Name (ac))
        |R1 ~= Race_Title (R))->
        Picked (CN, R1) = Picked' (CN,R1))
        & currentRace = R)
    & Signal_Enabled & Which_Signal = Vote_Signal
    & pickedName = Candidate_Name (vc)
    & attPickedName = Candidate_Name (ac)
    & Vote_Changed
```

The enabling condition for this threat action (i.e., for the transition) specifies that the fleeing voter is voting during election period in the voter's terminal mode, that there exists a race `R` such that the voter's candidate `vc` is in the displayed candidates list for race `R` for which the voter already voted, that the attacker's candidate `ac` is also a legitimate candidate contained in the displayed list for the same race `R` and it is not selected by the voter, and the voter's choice is different from the attacker's (i.e., `vc` ∼ = `ac`). In addition, the voter has already requested the review screen, currently there is nothing shown on the `REVIEW_SCREEN`, and there is no change of vote at the moment.

After the threat action is successfully executed (i.e., after the transition is ended) the following holds: the voter's selection in `tempVoteRecord` now contains the attacker's choice, the `Picked` value is *true* for `ac` and is `false` for voter candidate `vc`. In addition, the exported variables `currentRace`, `pickedName`, `attPickedName`, `pickedValue` and `Which_Signal` have new values, and the signaling variable is *true*. This indicates that the RTAL can now print the modification expressed in these exported variables. The RTAL process prints this information by executing the `Print_Selection` transition (see Section 5.1.2).

The above modification, which is contained in the `tempVoteRecord` variable, is also displayed on the review screen. It is worth noting that both the review screen and what is printed on the RTAL tape report the modified selection, rather than the original one. From an attacker's point of view, it is better to keep the display and tape consistent. The reason is that if an abnormality is detected, then it is more likely to be attributed to a display miscalibration rather than to an attack. It is possible that the voter will detect such a change. In this case, the voter can recast his/her vote by calling the `Push_Button` and `Make_Selection` transitions.

The `Attack_Change_Vote` also has exceptions that specify the other cases of the *voter type*

```
  EXCEPT
  [...]
  /*This attack assumes a Careful voter. The voter has confirmed and the Thank_You
    message has been displayed.*/
    Which_Phase = During_Voting
    & vType = Careful
    & EXISTS b: Button (b = CONFIRM
      & Button_Pushed (b))
    & scrName = THANKYOU_SCREEN
    & EXISTS R: Race (
      vc ISIN Displayed_Candidates (R)
      & ac ISIN Displayed_Candidates (R)
      & Picked (Candidate_Name (vc), Race_Title(R)))
    /*voter candidate is different from attacker candidate */
      & vc ~= ac
      & Min_Display (scrNumber)= Display_Info (Thank_You, NoButton)
      & ~Summary_Sent2RTAL
      & ~Vote_Changed & NormalVotingProcess
    EXIT
      EXISTS R: Race (vc ISIN Displayed_Candidates'(R)
      & ac ISIN Displayed_Candidates' (R)
      & vc ISIN tempVoteRecord' (R)->
        (Picked (Candidate_Name (vc), Race_Title(R))
          BECOMES FALSE
        & Picked (Candidate_Name (ac), Race_Title(R))
          BECOMES TRUE
        & currentRace = R))
  /*enabling RTAL to print the attacker's intention.*/
    & Signal_Enabled & Which_Signal = Vote_Signal
  /*for this candidate, print Cancelled on the RTAL tape.*/
    & pickedName = Candidate_Name (vc)
  /*for this candidate, print Selected on the RTAL tape.*/
    & attPickedName = Candidate_Name (ac)
    & Vote_Changed
```

The following snippet exception models the *complete voting process attack* (as shown in Fig. 4(a)). This attack assumes a fleeing voter who voted for the attacker's candidate, and hence completes the voting process.

```
  EXCEPT
  [...]
    & vType = Fleeing
    & scrName = REVIEW_SCREEN
    & scrNumber = Number_Of_Race + 1
    & Now - Change (scrNumber)>= 10 [...]
    & vc = ac
    & Review_Displayed
    & NormalVotingProcess
  EXIT
    /*the attacker calls the confirmation function and completes the process.*/
    scrNumber = scrNumber' + 2
    & scrName = THANKYOU_SCREEN
    & EXISTS b: Button (b = CONFIRM
      -> Button_Pushed(b) BECOMES TRUE)
    & Min_Display(scrNumber)
      BECOMES Display_Info(Thank_You, NoButton)
  /*The normal voting process is interrupted by the attacker and the DRE is not
    chirping for this voter.*/
    & ~NormalVotingProcess
```

The *Faking a Fleeing Voter* attack is an example of a scenario that requires several threat actions. The *canceling of votes* by faking a fleeing voter has three threat actions (as depicted in the sequence diagram, see Fig. 5). The threat actions are specified as three transitions in the DRE process:

1. *Attack_Change_Vote.* This is an except/exit transition that specifies the fact that an attacker fakes a fleeing voter by pretending to complete the voting process on her/his behalf. The exit assertion of this transition will set the variable `Fleeing_Faked` to *true*.
2. *Attack_ReDisplay.* This transition specifies the fact that after some delay (during which time the voter leaves the booth) since the voter is successfully fooled, the attacker directs the DRE to display the confirmation page again.
3. *Attack_Call_ChirpingR.* This specifies that after the voter has been fooled and *DelayTime* has passed, the attacker resumes the normal voting process by calling the chirping routine. This results in the poll worker taking action according to the prescribed procedure. (This transition is only enabled after the first two transitions have been executed.)

We say the *canceling of votes* is successful only after transition #3 has been executed. We omit the formal specifications for these threat actions, which are specified similarly to the others described earlier.

Table 1
Number of proof obligations and number of proved critical properties before and after the attack specification.

	Proof obligations	After splitting
	Invar, Constr, Sched	Invar, Constr, Sched
DRE	4, 6,1	10, 9, 2
RTAL	1, 1, 3	1, 1, 3
PEB	1, 0, 1	2, 0, 1
CFCard	0, 0, 1	0, 0, 2
Global	6, 0, NA	9, 0, NA
Total	12, 7, 6	22, 10, 8
Total proved, for the original specification		16,7,5
Total proved, after extending with attacks' information		6/16,2/7,0/5

7. Formal verification and results

We used the ASTRAL and PVS (Owre et al., 1993) tools to analyze whether the specification of the ES&S system meets the critical security requirements articulated previously. The main goal of our analysis is to provide the maximum assurance that the ES&S specification meets its critical requirements. To do that, it is necessary to generate proof obligations for critical requirements and prove them. ASTRAL supports two kinds of proof obligations: correctness proofs and consistency proofs. In the former case, the critical requirements of the system are proved to hold based on the executions of each process. In the latter case, it is proved that any assumptions made in the system are never false. Both proof obligations are useful for the e-voting systems in order to run a correct and secure election. In our verification, we focused on the correctness proofs of the ES&S voting system. We mainly attempt to prove the invariants and schedules clauses related to the components (i.e., DRE, RTAL, PEB, and CF Card) in isolation and the system as whole to hold at all times.

Before attempting the proofs with the PVS theorem prover, we should assure that the specification contains as few errors as possible, by performing a sequence of less costly steps. In ASTRAL, these steps include model checking and proof sketch construction. We applied the proof sketch construction strategies (such as proof ordering, transition steps, global and imported variable obligations) for our purpose. Before invoking the theorem prover, the ASTRAL split engine was used to split and classify the ASTRAL specification into collections of simpler properties that infer the whole clause so that the proof of each property could be tackled separately. The splitter can be invoked on any section of an ASTRAL specification that resolves to a boolean expression.

The specification of the ES&S system was first constructed and type-checked using the ASTRAL SDE. Thereafter, from the validated specification, we generated the corresponding proof obligations for the critical requirements. It is important to emphasize that the generated proof obligations are that of the intra-level proof obligations, which deal with proving that each process level satisfies its stated critical requirements and that the system level specification is consistent and satisfies the global requirements. Moreover, the specification was automatically translated into its PVS counterpart using the ASTRAL SDE, which enabled the specification to be passed to the PVS theorem prover for verification.

Table 1 shows the number of invariants, schedules, and constraints for each of the four processes and the global invariants. It also shows the number after they are split by the ASTRAL SDE for which we discharged the proof commands.

The assurance of the ES&S specification cannot be achieved without performing system proofs within the theorem prover. So far we managed to formally verify that the specification satisfies many of the critical requirements that we discussed previously, mostly the local invariants and constraints. The proofs were achieved by following the techniques presented in Kolano (1999). For instance, we applied the *try-untimed* and *try-untimed-con* proof strategies to prove some of the local invariants and constraints of the system.

As shown in Table 1, before extending the specification with attack information, we successfully proved 16 of the 22 invariants, 5 of the 8 schedules, and 7 of the 10 constraints. We expect that the other global and local properties can be proved using the same or similar proof techniques and strategies. After extending the system specification with the threat actions, we generated the proof obligations for the extended specification. However, unlike the original specification, there are more proofs to be done since there are additional transitions that correspond to the threat actions. In addition, because some of the original transitions were split and/or extended, the corresponding proof obligations must be reproved. In order to prove the requirements, we followed the same procedure as before. However, the proving process is very complex in the extended model. We started with the proof obligations that were unchanged to assure that they are still valid. So far, we have reproved 6 of the 16 invariants and 2 of the 7 constraints (see last row of Table 1).

Most of the reproved properties in the extended model are not complex. For instance, "The copy of the ballot images downloaded from the DRE must be equal to the ballot that was loaded into the DRE prior to start election;" (local DRE invariant property), "RTAL must scroll forward Min_ScrollForward_Position amount after the vote summary has printed;" (local RTAL constraint property), and "Each voter's choice will be printed after the vote signal is enabled and enough time has elapsed for the choice to be printed;" (local RTAL schedule property) easily reproved with additional proof strategies that are discharged to consider the added specification corresponding to the attack actions.

By analyzing these obligations in the extended model, we have attempted to understand why they were proved and why the others did not prove. We learned that, in most of the cases, to reprove these obligations we had to discharge a few more proof steps as compared to proving in the original specification. In the majority of the cases, however, the current ASTRAL specific proof strategies are not sufficient for completing the proof. We have also faced some resource limitations – namely, heap storage problem although we assigned the maximum heap size (we used 4G RAM) – for the PVS Allegro Lisp. These are some of the reasons that we did not complete the remaining proof obligations. Hence, there are more complex proof obligations which need more powerful proof strategies to complete the proof. Currently, we are working on modularly approaching the proof strategy and possibly extending the ASTRAL specific PVS strategies.

The main lessons we learned span from understanding the voting process followed in the USA, deriving the specification, up to the usage of PVS. We started the specification by looking at the video about how to vote using the ES&S system, the various documents about the system specification and machine usage scenario, and some known requirements recommendations. This allowed us to learn the various components, the underlying communication among them, and the kinds of data they exchange during the communication (e.g., DRE sends the "selection" information to the RTAL). Converting these concepts to a formal language is very complex and demands a clear understanding of the process, each component's behavior, their combined behaviors, and the properties/requirements, as well as the specification language itself. In fact, formalizing these using ASTRAL was relatively easy as the language is closer to higher-level language. However, the difficulty arose when using the PVS system. Currently, we are investigating how to modularize the proof strategy and possibly extend the ASTRAL specific PVS strategies. The ASTRAL language contains structuring mechanisms that allow one to build modularized specifications of complex systems. It is interesting to specify the attack scenarios in separate specifications and try to compose them using the composition mechanism of ASTRAL.

8. Related work

Scientific literature on e-voting is wide and multi-disciplinary. For the purpose of this work, we organize previous work in two areas: designing better e-voting systems using formal methods (high-level assurance) and assessing existing systems (low-level assurance).

Applying formal methods for e-voting. The trends in this area focus in three closely related directions: verifying cryptographic protocols (e.g., Juels et al., 2005; Backes et al., 2008; Simidchieva et al., 2008; Delaune et al., 2009; Cansell et al., 2007a; Küsters et al., 2010), system behavior (e.g., Cansell et al., 2007b; Sturton et al., 2009; Tiella et al., 2006; Gibson et al., 2010), and procedures (e.g., Weldemariam and Villafiorita, 2008; Bryl et al., 2009).

With respect to the first (i.e., verifying cryptographic protocols), Delaune et al. (2009) particularly present a framework for formal specification and verification of e-voting protocol properties (an earlier version of this work can be found (Kremer and Ryan, 2005)). These properties are vote-privacy, receipt-freeness, and coercion-resistance. Their work mainly focused on formally verifying the correctness of protocols with respect to these properties. The authors used applied π-calculus (Abadi and Fournet, 2001) for the formalization of the voting protocol and for the properties, which are to be analyzed against the protocol model using the ProVerif tool (Blanchet, 2009).

Juels et al. (2005) define a formal model for e-voting schemes that involves a more powerful adversary than previously proposed in the literature related to receipt-freeness – the inability of a voter to prove to an attacker that s/he voted in a particular manner, even if the voter wishes to do so. Their schemes allows an adversary to demand that coerced voters vote in a particular manner or that they disclose their secret keys. The authors also provide formal security definitions for essential properties of correctness, verifiability, and coercion-resistance. However, the paper did not consider a verification process using automated techniques.

The authors in Campanelli et al. (2008) used a CCS (calculus of communicating systems)-like process algebra with cryptographic primitives to specify and analyze some properties of the e-voting system they built. More specifically, they presented a small mobile implementation of an e-voting system named M-SEAS (Mobile Secure E-voting Applet System) and used formal verification techniques to validate the security properties of the system. Their analysis goal is checking whether their system is free from Sensus vulnerability[2] by using the Crypto-CCS language (Martinelli, 2002) and the PaMoChSA analysis tool.

With respect to the second focus (i.e., verifying system behavior), Simidchieva et al. (2008) demonstrate the usage of different technologies for specifying and verifying requirements for election processes – namely, by reasoning rigorously about the presence or absence of errors during all phases of an election process. In particular, they used the Little-JIL process definition language (Cass et al., 2000) to formally define election processes and the PROPEL tool (Smith et al., 2002) to support the development of precise lower-level properties, which are then fed to the verification system called FLAVERS (Cobleigh et al., 2002) to check whether the process model satisfies these properties. If the process model violates a property, the FLAVERS system provides counterexamples as traces. Traces are sequence of steps in the process model that may lead to the property violation as generated by the verification system. Such traces can then be used to guide the improvement of the process model. Similarly, Villafiorita et al. (2009); Tiella et al. (2006) demonstrate the integration of formal methods in the development process of a voting system named ProVotE. In particular, the authors specified the behaviors of voting control logic using UML finite state machine and developed a tool named FSMC+ (Tiella et al., 2007) that automatically generates NuSMV (Cimatti et al., 2002) code corresponding to the specified FSM (this helped for the structuring and managing requirements discussed in Weldemariam et al., 2009). Then they performed the verification using the NuSMV model checker. The results of the model checker, presented in the form of counterexample, are then analyzed. This enabled the authors to incorporate the analysis results of the verification into the actual development process of the core ProVotE system (Villafiorita et al., 2009).

Sturton et al. (2009) present an approach for the design and analysis of an e-voting machine based on a combination of formal verification and systematic testing. They formally verify the correctness of each of the individual components of a voting machine, as well as verifying some crucial correctness properties of their composition. Their work is targeted to the following verification goals: ensuring that each individual component of the voting machine and their composition should meet the specification of the individual components and their composition respectively; the voting machine should be structured to enable sound systematic system testing; ensuring that the voting machine must behave and store votes according to the voters selection when configured with a particular election definition file. For each module, they construct a formal specification that fully characterizes the intended behavior of that component. A number of properties related to the structural and functional aspects that the machine should satisfy are identified and specified. They used Verilog (Thomas and Moorby, 1991) for the implementation of their specification and the SMV analysis tool and satisfiability solving to verify that their Verilog implementation meets the specifications.

Moreover, Cansell et al. (2007b) attempted to use the B-method (Abrial, 1996) – a method for specifying, designing and coding software systems – to construct a formal, mathematical model of the e-voting problem, with the aim of demonstrating the use of formal methods for supporting the correct design and implementation of safe e-voting systems. More recently, Gibson et al. (2010) clearly illustrate the importance of formal software engineering in the development of e-voting systems. Interesting, the authors demonstrated the development of an e-voting system using different modeling languages to address different types of critical requirements.

Although there is limited research in this area, we have mentioned some existing works that use formal methods such as model checking and theorem provers to provide higher assurances for the design and implementation of e-voting systems. However, none of these works focus on the aspects related to procedures in their modeling and analysis (the third verification focus). In that regard, in Weldemariam and Villafiorita (2008) we complement such works by widening their scope of analysis with procedures analysis. An approach to reason on security properties of the to-be models (which are derived from as-is model) in order to evaluate procedural alternatives in e-voting systems is presented in Bryl et al. (2009). Additionally, the authors in Grimm et al. (2010) presented a formal model for the correction and abort requirement of e-voting with some concepts borrowed from Protection Profile (Volkamer and Krimmer, 2007) of the Common Criteria (Common and Criteria, 2007). More specifically, they first described a formal IT security model that allows the formalization of (some) basic security requirements for e-voting. Secondly, they modeled the corresponding security properties as secure system states using the same machinery. Thirdly, they specified state transition rules that control the voting behaviors. Finally, an attempt to mathematically

[2] In Sensus protocol (Cranor and Cytron, 1997), "this vulnerability basically allows one of the entities involved in the election process to cast votes of eligible users that, although registered, abstain to vote."

prove that functions following the rules would transfer a secure state into a secure state.

Assessing exiting e-voting systems. Some e-voting systems currently deployed in elections have undergone a thorough and independent scrutiny to evaluate their security and quality. Security vulnerabilities have been reported in each aspect of security – that is, technological, socio-technical, and social aspects. These vulnerabilities have been practically investigated and proved by various academic researchers. This creates an enigma in the trustworthiness of the machine and the voting process as well (Bishop and Wagner, 2007; Bryans et al., 2006).

In line with this, we mention the following academic research (Kohno et al., 2004; Aviv et al., 2008; Gardner et al., 2007; Balzarotti et al., 2008; Jones, 2003; Ansari et al., 2008). These works assess both hardware and software of different forms of e-voting machines (e.g., Diebold/Premier, ES&S, InterCivic), which are mostly in use in some 37 U.S. states. The studies identified serious design and implementation flaws, which are notable for their level of egregiousness. More specifically, their analysis have showed that the current e-voting systems are vulnerable to very serious attacks, and they have produced a catalogue of vulnerabilities and possible attacks. Some analyzes also suggested a drastic change in the way in which e-voting systems are designed, developed, and tested (e.g., by identifying procedures to eliminate or mitigate the discovered issues, by developing a precise methodology and toolsets for the assessment). The assessment methodology presented in Balzarotti et al. (2010) is particularly astonishing as it provides various insights on each individual and in-depth step of the analysis, to be reparable also. It can be used for other complex-security critical systems evaluation and assessment as well as to the software testing community.

On top of the above technical security evaluations (i.e., the evaluation of the source code of the machines), works such as Volkamer (2009) and Schmidt et al. (2010) focus on the definition of common standards and methodologies for evaluation and certification for e-voting systems. Some works on going with respect to development of common standards along the CC Protection Profile. This provides a basis for standardized evaluations with comparable results. Currently, there are two Protection Profiles, one for remote electronic voting and one for the digital election pen(see in Volkamer, 2009). The latter one has been used for a real system and the first one is applied to the Polyas system (Reinhard and Jung, 2007). Schmidt et al. (2010) also discuss an evaluation and certification approach for Voting Service Providers (VSP), by combining the evaluation of remote e-voting system and operational environment. The VSP is required to provide a security concept in which it demonstrates satisfaction of the security requirements defined in the legal regulation. The authors suggested the incorporation of existing evaluation methodologies to facilitate the evaluation and certification process.

9. Conclusions and future work

Electronic voting is about the behavior of all the voting components (be they electronic, mechanic or human), and so assurance of electronic elections requires one to investigate all these aspects in an integrated way. A robust design means that a system continues to function even when one or more components is compromised. Robust designs are usually achieved by redundancy. Electoral systems are redundant: votes are stored electronically and on paper, poll workers control the behavior of machines and the machines limit in some way the operations poll workers can perform (e.g., by logging their activities). However, a robust redundant architecture requires one to clearly allocate responsibilities and priorities. This can only be achieved by an integrated analysis of the voting scenarios and by a clear allocation of the requirements for the different "components" of an election system.

In this paper, we have shown how formal verification techniques can be used to model and reason about the security of e-voting systems. Our approach consists of formulating each individual component of the voting system as a process instance in ASTRAL and the specification and verification of critical security properties about individual components and about the system as a whole. We also specified the attack scenarios that are reported by academic studies on the security testing and analysis of such systems. Namely, along the line of nominal behavior specification, we model and specify attacks. We extend the specifications that describe the nominal behaviors of the system under analysis by augmenting them with the attack model. Thereafter, we attempted to analyze the extended model against the same set of critical requirements as the nominal model should meet. The threat actions that we specified in the extended system were those needed to model the specific scenarios presented in McDaniel et al. (2007).

Although our approach provides certain benefits over existing work in the area, it is in no way a verification "silver bullet". As with any formal verification technique, it requires the use of formal languages, various analytic tools including a theorem-proving system, and considerable skill in understanding the various information sources as well as understanding various components of the system. Our work is not a substitute for existing approaches to the assessment and verification of specific components of the voting system. Rather, it complements them by reasoning about the interaction of "components" (human and technologies). This is analogous to integration and unit testing, which are different in scope, but both are necessary to ensure a system's correctness. We must also be clear that this research work does not consist of a novel specification technique nor a novel voting system.

Additionally, this work allowed us to learn the operational scenario for various e-voting components, the underlying communication among them, and the kinds of data they exchange during the communication. Converting these concepts to a formal language is very complex and demands a clear understanding of the process, each component's behavior, their combined behaviors, and the properties/requirements, as well as the specification language itself. In fact formalizing these using ASTRAL was relatively easy as the language is closer to higher-level language. We acknowledge, however, that some issues remain open. One is the complexity of some proofs, for which more powerful strategies or interactive support could be helpful. The other is modeling other kinds of attacks with particular reference to those cutting across the structure of the original requirements model. These make incremental or compositional verification challenging. Our paper does not provide a solution to this problem, but it does provide further evidence that the problem is significant and worthy of further research. However, this work demonstrates how formal methods can be used for the specification and verification of e-voting systems in order to guarantee the correctness of the system. The success of the next generation of e-voting machines depends upon being able to capitalize on the lessons learned from different disciplines. The work we have presented in this paper is one way in which we can get a better understanding of the strengths and the weaknesses of existing techniques and thus lay the foundations for engineering, designing, implementing, as well as deploying a new generation of more secure and robust technologies for polling stations.

Acknowledgments

This work was started while Komminist Weldemariam was visiting the Security Group at the University of California at Santa Barbara. The authors would like to thank the anonymous reviewers who gave many useful suggestions and comments on the first version of this paper.

References

Abadi, M., Fournet, C., 2001. Mobile values new names and secure communication. SIGPLAN Notices 36 (3), 104–115.

Abrial, J.-R., 1996. The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA, ISBN 0-521-49619-5.

Ansari, N., Sakarindr, P., Haghani, E., Zhang, C., Jain, A.K., Shi, Y.Q., 2008. Evaluating electronic voting systems equipped with voter-verified paper records. IEEE Security and Privacy 6 (3), 30–39.

Aviv, A., Černy, P., Clark, S., Cronin, E., Shah, G., Sherr, M., Blaze, M., 2008. Security evaluation of ES&S voting machines and election management system. In: EVT: Proceedings of the Conference on Electronic Voting Technology , USENIX Association, Berkeley, CA, USA, pp. 1–13.

Backes, M., Hritcu, C., Maffei, M., 2008. Automated verification of remote electronic voting protocols in the applied pi-calculus. In: CSF, IEEE, ISBN 978-0-7695-3182-3, pp. 195–209.

Balzarotti, D., Banks, G., Cova, M., Felmetsger, V., Kemmerer, R.A., Robertson, W.K., Valeur, F., Vigna, G., 2010. An experience in testing the security of real-world electronic voting systems. IEEE Transaction on Software Engineering 36 (4), 453–473.

Balzarotti, D., Banks, G., Cova, M., Felmetsger, V., Kemmerer, R., Robertson, W., Valeur, F., Vigna, G., 2008. Are your votes really counted? Testing the security of real-world electronic voting systems. In: ISSTA: International Symposium on Software Testing and Analysis ,. ACM, New York, NY, USA, pp. 237–248.

Bishop, M., Wagner, D., 2007. Risks of e-voting. Communication of ACM 50 (11), 120–1120.

Blanchet, B., 2009. Automatic verification of correspondences for security protocols. Journal of Computer Security, 363–434.

Bryans, J.W., Littlewood, B., Ryan, P.Y.A., Strigini, L., 2006. E-voting: dependability requirements and design for dependability. In: ARES ,. IEEE Computer Society, Washington, DC, USA, pp. 988–995.

Bryl, V., Dalpiaz, F., Ferrario, R., Mattioli, A., Villafiorita, A., 2009. Evaluating procedural alternatives: a case study in e-voting. EG 6 (2), 213–231.

California Secretary of State, 2007. Withdrawal of Approval of Diebold Election Systems, Inc., GEMS 1.18.24/AccuVote-TSWAccuVote-OS DRE & Optical Scan Voting System and conditional re-approval of use of Diebold Election Systems, Inc., GEMS 1.18.24/AccuVote-TSX/AccuVote-OS DRE & optical scan voting system, URL http://www.sos.ca.gov/voting-systems/oversight/ttbr/diebold-102507.pdf.

Campanelli, S., Falleni, A., Martinelli, F., Petrocchi, M., Vaccarelli, A., 2008. Mobile Implementation and Formal Verification of an e-Voting System. In: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services, IEEE Computer Society , Washington, DC, USA, pp. 476–481.

Cansell, D., Gibson, J.P., Mery, D., 2007a. Formal verification of tamper-evident storage for E-voting. In: SEFM, IEEE Computer Society , Washington, DC, USA, pp. 329–338.

Cansell, D., Gibson, J.P., Méry, D., 2007b. Refinement: a constructive approach to formal software design for a secure e-voting interface. Electronic Notes in Theoretical Computer Science 183, 39–55.

Cass, A.G., Lerner, B.S., Sutton, S.M., McCall, E.K., Wise, A., Osterweil Jr., L.J., 2000. Little-JIL/Juliette: A process definition language and interpreter. In: Proceedings of the 22nd International Conference on Software Engineering , ICSE'00, ACM, New York, NY, USA, pp. 754–757.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A., 2002. NuSMV 2: an open source tool for symbolic model checking. In: Computer Aided Verification, Lecture Notes in Computer Science ,. Springer, Berlin, Heidelberg, pp. 241–268.

Cobleigh, J.M., Clarke, L.A., Osterweil, L.J., 2002. FLAVERS: a finite state verification technique for software systems. IBM Systems Journal 41 (1), 140–165, ISSN 0018-8670.

Common Criteria Common Criteria for Information Technology Security Evaluation, 2007. http://www.commoncriteriaportal.org/.

Council of Europe Recommendation on legal, Operational and Technical Standards for e-voting, Council of Europe, 2004.

Cranor, L.F., 1996. Electronic voting: computerized polls may save money protect privacy. Crossroads 2 (4), 12–16, ISSN 1528-4972.

Cranor, L.F., Cytron, R.K., 1997. Sensus: a security-conscious electronic polling system for the Internet. Hawaii International Conference on System Sciences 3, 561, ISSN 1060-3425.

Delaune, S., Kremer, S., Ryan, M., 2009. Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17 (4), 435–487.

Election Systems & Software, Video: Voting with the ES&S iVotronic, 2009. http://www.essvote.com/HTML/video/ESS_IVO.wmv.

Federal Election Commission, Voting System Standards, 2005. URL http://www.eac.gov/.

Gardner, R., Garera, S., Rubin, A., 2007. On the Difficulty of Validating Voting Machine Software with Software. In: EVT: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop ,. USENIX Association, Berkeley, CA, USA.

Gibson, J.P., Lallet, E., Raffy, J.-L., 2010. Engineering a distributed e-voting system architecture: meeting critical requirements. In: ISARCS, pp. 89–108.

Grimm, R., Hupf, K., Volkamer, M., 2010. A formal IT-security model for the correction and abort requirement of electronic voting. In: Electronic Voting , pp. 89–107.

Heitmeyer, C.L., Archer, M.M., Leonard, E.I., McLean, J.D., 2008. Applying formal methods to a certifiably secure software system. IEEE Transactions on Software Engineering (1), 34, pp. 82–98.

ES&S Inc, Election Systems & Software: iVotronic^{TM}Voting System, version 9.1.x Election Day Operations Checklist, 2007.

Jones, D.W., 2003. The Evaluation of Voting Technology, Chap 1. Advances in Information Security. Kluwer Academic, pp. 3–16.

Juels, A., Catalano, D., Jakobsson, M., 2005. Coercion-resistant electronic elections. In: WPES: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society ,. ACM, New York, NY, USA, pp. 61–70.

Küsters, R., Truderung, T., Vogt, A., 2010. Proving coercion-resistance of scantegrity II , ICICS, Lecture Notes in Computer Science. Springer, pp. 281–295.

Kemmerer, R.A., 1990. Integrating Formal Methods into the Development Process. IEEE Software 7 (5), 37–50.

Kohno, T., Stubblefield, A., Rubin, A.D., Wallach, D.S., 2004. Analysis of an electronic voting system. In: IEEE Symposium on Security and Privacy , p. 27.

Kolano, P., 1999. Tools and Techniques for the Design and Systematic Analysis of Real-Time Systems. Ph.D. thesis, University of California, Santa Barbara).

Kolano, P.Z., Dang, Z., Kemmerer, R.A., 1999. The Design and Analysis of Real-Time Systems Using the ASTRAL Software Development Environment. Annals of Software Engineering 7 (1–4), 177–210.

Kremer, S., Ryan, M.D., 2005. Analysis of an electronic voting protocol in the applied Pi-calculus. In: Sagiv, M. (Ed.), Programming Languages and Systems—Proceedings of the 14th European Symposium on Programming (ESOP'05). Lecture Notes in Computer Science, Springer, Edinburgh, UK, pp. 186–200.

Lowry, M., Dvorak, D., 1998. Analytic Verification of Flight Software. IEEE Intelligent Systems 13 (5), 45–49.

Martinelli, F., 2002. Symbolic semantics and analysis for crypto-CCS with (almost) generic inference systems. In: MFCS: Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science ,. Springer-Verlag, London, UK, pp. 519–531.

McDaniel, P., Butler, K., Enck, W., Hursti, H., McLaughlin, S., Traynor, P., Blaze, M., Aviv, A., Cerny, P., Clark, S., Vigna, G., Kemmerer, R., Balzarotti, D., Banks, G., Cova, M., Felmetsger, V., Robertson, W., Valeur, F., Hall, J.L., Quilter, L., 2007. EVEREST: Evaluation and Validation of Election-Related Equipment, Standards and Testing, Ohio Secretary of State's EVEREST Project Report.

McGaley, M.,2008. E-voting: an Immature Technology in a Critical Context. Ph.D. thesis, Departement of Computer Science, National University of Ireland, Maynooth, URL http://eprints.nuim.ie/1486/.

Mercuri, R.T., 2001. Electronic Vote Tabulation Checks and Balances. Ph.D. thesis, University of Pennsylvania.

Owre, S., Shankar, N., Rushby, J.M., 1993. The PVS Specification Language.

Reinhard, K., Jung, W., 2007. Compliance of POLYAS with the BSI Protection Profile—Basic Requirements for Remote Electronic Voting Systems. In: VOTE-ID ,. Springer, pp. 62–75.

Sastry, N.K., 2007. Verifying Security Properties in Electronic Voting Machines, Ph.D. thesis, EECS Department, University of California, Berkeley, URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-61.html.

Schmidt, A., Volkamer, M., Buchmann, J., 2010. An Evaluation and Certification Approach to Enable Voting Service Providers. In: Electronic Voting , pp. 135–148.

Simidchieva, B.I., Marzilli, M.S., Clarke, L.A., Osterweil, L.J., 2008. Specifying and verifying requirements for election processes. In: Proceedings of the 2008 International Conference on Digital Government Research , Digital Government Society of North America, pp. 63–72.

Smith, R.L., Avrunin, G.S., Clarke, L.A., Osterweil, L.J., 2002. PROPEL: an approach supporting property elucidation. In: ICSE'02: Proceedings of the 24th International Conference on Software Engineering ,. ACM, New York, NY, USA, pp. 11–21.

Sturton, C., Jha, S., Seshia, S.A., Wagner, D., 2009. On voting machine design for verification and testability. In: Al-Shaer, E., Jha, S., Keromytis, A.D. (Eds.), ACM Conference on Computer and Communications Security. , pp. 463–476.

Thomas, D.E., Moorby, P.R., 1991. The VERILOG Hardware Description Language. Kluwer Academic Publishers, Norwell, MA, USA.

Tiella, R., Villafiorita, A., Tomasi, S., 2006. Specification of the Control Logic of an eVoting System in UML: the ProVotE experience. Csduml, 84–94.

Tiella, R., Villafiorita, A., Tomasi, S., 2007. FSMC+ a tool for the generation of Java code from statecharts. In: PPPJ: Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java ,. ACM, New York, NY, USA, pp. 93–102.

Villafiorita, A., Weldemariam, K., Tiella, R., 2009. Development, formal verification, and evaluation of an E-voting System with VVPAT. IEEE Transactions on Information Forensics and Security 4 (4), 651–661.

Volkamer, M., 2009. Evaluation of Electronic Voting: Requirements and Evaluation Procedures to Support Responsible Election Authorities. Springer, ISBN 3642016618, 9783642016615.

Volkamer, M., Krimmer, R., 2007. Independent audits of remote electronic voting—developing a common criteria protection profile. In: Proceedings der EDEM 2007—Elektronische Demokratie in Österreich.

Volkamer, M., McGaley, M., 2007. Requirements and evaluation procedures for eVoting. In: ARES, IEEE Computer Society , pp. 895–902.

Weldemariam, K., Villafiorita, A., 2008. Modeling and analysis of procedural security in (e)voting: the Trentino's approach and experiences. In: EVT: Proceedings of the Conference on Electronic Voting Technology , USENIX Association, Berkeley, CA, USA.

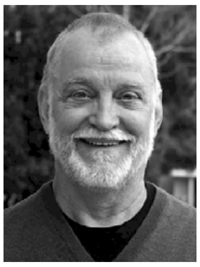
Weldemariam, K., Mattioli, A., Villafiorita, A., 2009. Managing Requirements for E-Voting Systems: Issues and Approaches. In: Proceedings of the 2009 First

International Workshop on Requirements Engineering for e-Voting Systems , IEEE Computer Society, Washington, DC, USA, pp. 29–37.

Wolchok, S., Wustrow, E., Halderman, J.A., Prasad, H.K., Kankipati, A., Sakhamuri, S.K., Yagati, V., Gonggrijp, R., 2010. Security analysis of India's electronic voting machines. In: Proceedings of the 17th ACM Conference on Computer and Communications Security , CCS'10, ACM, New York, NY, USA, pp. 1–14.

Xu, D., Nygard, K., 2005. A threat-driven approach to modeling and verifying secure software. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering , ASE'05, ACM, New York, NY, USA, pp. 342–346.

Komminist Weldemariam received the BS degree in Computer Science from Addis Ababa University, Ethiopia, in 2003, the MS degree in Computer Science and Engineering from Indian Institute of Technology, Bombay, India, in 2006, and the PhD degree in Computer Science from the University of Trento, Italy, in 2010. He is currently a post-doctorate follow in the Center of Information Technology at the Fondazione Bruno Kessler, Italy. He has been visiting scholar at the computer security group of the University of California, Santa Barbara. His research interests include Software Engineering, Security, Electronic Voting Systems, and ICT4G. He is a member of the IEEE.



Richard A. Kemmerer is the computer science leadership professor and a past department chair of the Department of Computer Science at the University of California, Santa Barbara. His research interests include formal specification and verification of systems, computer system security and reliability, programming and specification language design, and software engineering. Kemmerer has a PhD in Computer Science from the University of California, Los Angeles. He is a fellow of the IEEE Computer Society and of the ACM, and is a member of the IFIP Working Group 11.3 on Database Security and of the International Association for Cryptologic Research.

Adolfo Villafiorita received the MS degree from the University of Genoa, Italy, in 1993, and the PhD degree from the University of Ancona, in 1997, both in Computer Science. He is a senior researcher in the Center of Information Technology at the Fondazione Bruno Kessler, Italy. His current interests include ICT4G, software and system engineering, security, formal methods, and safety analysis. He has participated and led several industrial projects related to the development of safety critical applications in the railway, aerospace, and e-Government sector. He is a member of IEEE and ACM.