Development, Formal Verification, and Evaluation of an E-Voting System With VVPAT

Adolfo Villafiorita, Komminist Weldemariam, *Student Member, IEEE*, and Roberto Tiella

***Abstract*—The use of new technologies to support voting has been and is the subject of great debate. Several people advocate the benefits it can bring—such as improved speed and accuracy in counting, accessibility, voting from home—and as many are concerned with the risks it poses, such as unequal access (digital divide), violation to secrecy and anonymity, alteration of the results of an election (because of malicious attacks, bad design/coding, or procedural weaknesses). The attitude of different governments towards electronic voting (e-voting) varies accordingly. In this paper, we present the activities related to the development and formal verification of an e-voting system, called ProVotE. ProVotE is an end-to-end e-voting system with a voter verified paper audit trial, developed within the framework of a larger initiative whose goal is assessing the feasibility of introducing e-voting in the Autonomous Province of Trento. ProVotE has been used in trials and elections with legal value in Italy. What we believe to be of interest is the approach we took for its development, which has been based on a participatory design for the definition of the voter interface, on the usage of formal methods and model checking for the validation of the core logic of the machine, on open source components, and on the formal analysis of some critical procedures related to the usage of the machine during the election.**

***Index Terms*—Development, electronic voting (e-voting), formal specification and verification, ProVotE, security assessment.**

I. Introduction

VOTING systems are used in environmental conditions that are quite peculiar, if not unique. In fact, they are probably the only safety and security critical systems for which all these conditions are met: they run in an environment with limited control; they are operated by people with the most diverse training and experience; they have to be accurate, while, at the same time, have stringent requirements related to what they can trace, to protect secrecy and anonymity. Logistics and support, due to the geographical distribution and time constraints of elections, add further complexity. The development and verification of fair and secure e-voting machines, therefore, not only has to take into account their inherent features (e.g., what kind of protection they offer to tampering with), but also how they fit into the environment in which they are used and in the electoral process (as also suggested, e.g., in [1]–[5]).

We are interested in digital recording electronics with printed audit trails (DRE-VVPAT), that is, touchscreen-based machines that produce a printout of each vote, verified directly by the voter, to maintain a physical and verifiable record of the votes cast.

ProVotE is a project sponsored by the Autonomous Province of Trento (Italy) with the goal of evaluating the switch to electronic systems for local elections. During the project, we decided to develop our own DRE-VVPAT system. This choice was mainly driven by the possibility of incorporating feedback from voters (and improve usability), guiding the design based on other experiences and recommendations (e.g., [6]–[9]), and choosing technologies and components. On top of that, it allowed us to adopt formal methods for the development of some critical components.

The size of the screen and of the printed audit trail, a signaling system to display the status of the machine to the poll workers, the programming language and the OS—Java and Linux, respectively—are some distinguishing features of the resulting machine.

The ProVotE machine has been used with experimental value in various trials during local elections and in two small elections with legal value.[1] The trials allowed to collect plenty of data about the machines and feedback from the citizens, as illustrated in, e.g., [10] and [11]. The number of people involved by the trials (about 28 000) and the scope of the project make ProVotE the biggest e-voting initiative in Italy.

This paper presents some of the key development activities of the e-voting machine, that we believe to be of general interest and applicability to other contexts and domains. Some of the work described in this paper revises and details works of the authors published elsewhere (see [12]–[15]).

This paper is structured as follows. In Section II, we give a brief overview of voting in Italy and about the scope of the technological initiatives within the ProVotE project. Section III introduces the ProVotE e-voting machine along with the key components of the system. Section IV presents the development approach we adopted. Sections V and VI discuss the use of formal specification and validation techniques for the analysis of procedural security and for the development of the control logic of the voting machine. In Section VII we discuss some related work. Finally, Section VIII draws some conclusions and some possible future directions.

Manuscript received February 14, 2009; revised September 30, 2009. First published October 20, 2009; current version published November 18, 2009. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Aviel D. Rubin.

A. Villafiorita and R. Tiella are with the Center for Scientific and Technological Research (IRST) Foundation Bruno Kessler (FBK), Trento 38050, Italy (e-mail: adolfo@fbk.eu; tiella@fbk.eu).

K. Weldemairam is with the Department of Information Engineering and Computer Science, University of Trento, Trento 38050, Italy (e-mail: weldemar@disi.unitn.it).

Digital Object Identifier 10.1109/TIFS.2009.2034903

[1]One is the election of the students' representatives of a High School, in Italy regulated by law; the other a local poll to unite two municipalities in Regione Friuli Venezia Giulia.

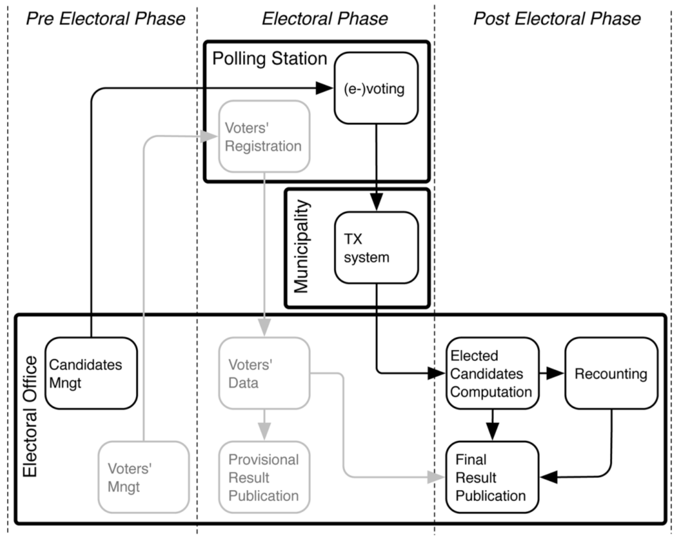

Fig. 1. Elections in Italy and the ProVotE system architecture.

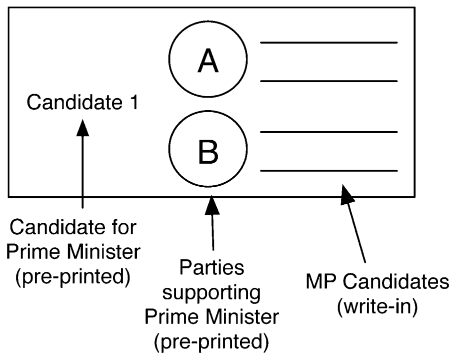
A
B
Candidate 1
Candidate for Prime Minister (pre-printed)
Parties supporting Prime Minister (pre-printed)
MP Candidates (write-in)

Fig. 2. Basic element of a paper ballot.

II. Overview of Voting in Italy

Elections differ quite a bit from nation to nation, not only with respect to the system chosen to determine the elected candidates (e.g., proportional, majoritarian), but also for the procedures, the way in which votes can be cast, the organizations involved, etc. Looking at Europe, we range from cases such as that of Estonia, in which voters can cast their vote by mail, through the internet, or by going to the polling station, to that of countries, such as Italy, in which the voting procedures are completely manual and voters are assigned a specific polling station to cast their vote.[2]

In this paper, we focus on the procedures in Italy for which we provide a very high level view in Fig. 1. The picture also allows us to highlight the targets of automation of the ProVotE project.

The dotted lines represent different phases of the voting process. We distinguish, in particular, three phases: pre-electoral (lasting up to six months before the election), electoral (in Italy lasting for one or two consecutive days), and post-electoral (lasting from a few days up to months, in case of litigations or irregularities). The boxes in bold lines represent the organizations responsible for the process. In particular, the Electoral Office, responsible for the overall operations, the Polling Stations, where votes are cast, and the Municipality, coordinating some operations at the local level. The picture abstracts away various other actors that participate in the process, including Tribunal, Ministries, Police forces, and private contractors (e.g., the firms responsible of printing the paper ballots). Finally, the rounded rectangles represent functions performed during an election. There are two main chains of functions, voters' management and votes' management.

The first chain, voter management, is pictured in gray and it includes all the operations and data for making sure that only the people with a right are given access to voting and that each citizen can cast at most one vote. Access to voting is based on the maintenance of redundant copies of paper books containing the list of people who can vote in each polling station. Such books, kept and updated between elections in central offices, are distributed to the polling stations before the election. During the election, at designated times, data about participation to the election is communicated (via fax or phone) to the Electoral Office, which makes the aggregated provisional data available to the Internet via a web system. At the end of election, the final data about participation is used to verify the correspondence between number of votes and voters.

The second chain, vote management, includes all operations related to managing votes. The Electoral Office is responsible for collecting names of the main candidates (e.g., people contesting for the prime minister position), the list of parties supporting each candidate, and the names of the candidates of each party (e.g., people contesting as MPs). The information determines the layout of the printed ballots. A cross can be used to express a preference for a party or a Prime Minister candidate. To vote for an MP, the voter writes the name of the MP close to the party to which the MP belongs. Simplifying the law quite a bit, misspelling or writing the name of the candidate in the wrong position causes the ballot to be void. See Fig. 2 for an example of the layout of a ballot.

Votes are collected in the polling stations, where they are kept until the end of the election day, when the poll worker initiates tabulation of data. Registers with the results and all the ballots are then returned to the Electoral Office, which aggregates the results and makes provisional results available. To speed things up, *provisional* results are also transmitted by poll workers from the Municipalities to the Electoral Office using a web-based system.

Registers and ballots are then analyzed by the Electoral Office. If irregularities are found, appropriate actions, such as recounting, are taken. The results are then declared final.

Short of the transmission of provisional results, all other functions are performed manually. This provides various opportunities for automation. In the case of the ProVotE project, we mainly focused on e-voting and on systems to automate the recounting of the ballots produced by the e-voting machines. The voting machine is described in this paper. Concerning the recounting system, we just mention that the application is operated by poll workers, who use a barcode reader to read the barcodes printed on the ballots. The application reproduces on screen the ballot corresponding the barcode just read and updates the total

[2]In the Italian case, postal voting has been introduced recently and only for Italian citizens residing in other countries.

tally. The system, however, is not meant as a replacement of manual recounting.

The delivery of election results has varied from physical transportation to the Electoral Office to electronic upload through a web application. Concerning the systems for determining the elected candidates, we tried both a commercial system adopted by various Electoral Services in Italy and an in-house solution. Finally in one trial we also tested a system for automating registration of voters.

Differently from what happens in the paper-case, we always recounted the printed ballots produced by the e-voting machines, to check consistency of data. To verify also the recounting application, in some cases, we also performed manual recounts of the ballots.

III. ProVotE E-Voting Machine: An Overview

A. Machine Components

We experimented with three different hardware prototypes of the e-voting machine: two monolithic designs—in which all components are enclosed in a single case—and one nonmonolithic design, in which DRE, VVPAT, and uninterruptible power supply (UPS) are three separated components, assembled in the polling station. The nonmonolithic design, although simpler to handle, raised concerns related to accidental or malicious unplugging of the cables connecting the components. The connection between the UPS and printer resulted particularly critical, because the printers we use do not have batteries. Any loss of connection to the UPS, although unlikely, thus, might have caused an interruption of the printing operations, leaving a half-printed ballot pending on the printer. Protecting anonymity in such situations would have been quite hard.

We decided, therefore, to stick with the monolithic design, which is installed in a standard electoral cabin (to protect privacy) and whose components are:

1) *Touchscreen*. Used to administer the machine and cast votes. It is a 20-in-wide screen, to improve readability.
2) *Thermal printer*. Protected behind a glass. It has a paper roll which is 12 in wide, to improve readability, and it is equipped with a cutter. Each vote, once it has been confirmed or rejected by the voter, is cut and falls in a ballot box, installed in the machine, below the printer. This protects from possible breaches to anonymity due to the possibility of tracing the order in which votes have been cast, as it might happen with VVPAT with a continuous tape.
3) *A UPS*. It provides emergency power, in case of power outage, to complete any pending voting operation before halting. The machine is not meant to be used in case of a long lasting power outage.
4) *A computer*. The computer (a standard PC) controls all the components of the voting machine.

Finally, two components of the machine, connected via cables to the main body of the machine, are installed on the desk of the poll worker:[3]

[3]The connectors are inside the case and secured, so that they cannot be accidentally or voluntarily unplugged without opening the machine. The choice of installing the smartcard reader externally to the cabin was made to prevent attacks such as the usage of maliciously crafted smartcards.

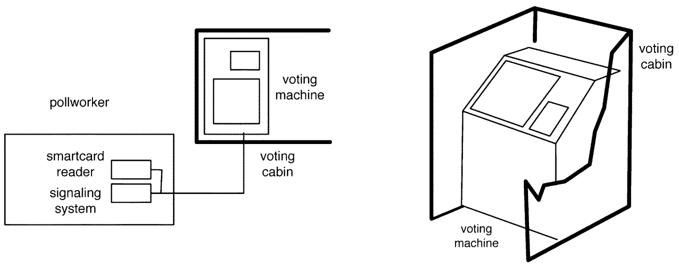

Fig. 3. ProVotE voting machine.

1) *A smartcard reader*. To control access to the machine. We used standard off-the-shelf smartcard technology.
2) *An external control display*. To show information on the voting process, such as the number of voters, the status of the machine (idle or busy), and the error conditions (e.g., printer nearing end of paper).

The cables are six meters long. This allows to position the machine and the "external" components so that the privacy of the voter is guaranteed (see Fig. 3).

Access to the critical parts of the machine (e.g., PC, ballot box) is protected by physical locks, whose keys are the responsibility of the poll workers.

B. Machine Configuration and Installation

The machines are delivered to the polling stations the day before the election, configured with the operating system and a bootstrapping application. We usually deliver two/three machines per polling station, according to the number of registered voters. Notice that, in Italy, no more than one thousand voters can be enlisted in a polling station.

The operating system is a stripped-down version of Linux, from which we removed all the components and daemons that are not strictly necessary to run the machine. The choice of Linux allows us to achieve two goals: access to the source code of the OS and the possibility of customizing the operating system.

The bootstrapping application is a custom application responsible of loading the voting software from an external device, verifying that the software is digitally signed by the Electoral Office, deciphering the software using a symmetric key stored in the OS, and launching the voting software (see below for more details).

Access to the operating system and the boot application is protected by physical means (keys to open the machine) and software configuration: booting order of devices, automatic launch of bootstrapping application, and password protection of BIOS and OS.

The voting software is distributed separately in a removable device (we used USB flash devices), shipped in a sealed envelope, with all the other materials necessary to run a machine (what we call the *voting kit*). There is one voting kit per machine.

The removable device contains the following.

1) A keyring, which stores:
   a) a private key (K_p) used by the machine to sign the data it produces;

b) a public key (K_u) used by the machine to verify the authenticity of the configuration data;

c) a secret key (K_s) for symmetric encryption of data.

The digital keys are specific to each voting kit (each kit has a different set) and protected by a PIN, known only by the poll worker.

2) Java byte-code of the e-voting application, digitally signed and encrypted with a symmetric key, stored in the OS.
3) Ballot configuration data, in an XML format, and symbols digitally signed and verifiable by K_u.
4) Other configuration data (e.g., localized user interface's textual messages, etc.), digitally signed and verifiable by K_u.
5) A file containing authentication credentials, encrypted using K_s. We use triple DES and SHA1 digests for crypting and signing.

The voting kit includes also three smartcards: 1) a smartcard to access the machine's administration functions and 2) two voting smartcards to enable the machine to cast votes. The smartcards are bound to the voting kit and can operate only the machine in which the voting kit is inserted.

The installation of the machine is performed by inserting the USB key into the machine (the USB port is inside the machine so that insertion and extraction require to use appropriate physical keys to open the machine). The bootstrapping application, after all the checks described above, copies the voting application into the internal hard disk and launches it. From now on the machine and the USB key are bound and there is no possibility of operating the machine with a different voting kit. Attempts to use a different USB key result in the system not booting.

C. Using the Machine

The ProVotE voting machine runs in two modes: in *poll worker mode* (administration) and *voter mode*. In the *poll worker mode*, the poll worker manages the voting machine through the machine life-cycle, which is encoded with a state machine, shown in Fig. 4.

Changes of state are permanent and irreversible: the machine starts in the last state reached and, once the poll worker confirms a change of state, it cannot be reversed. As a consequence, machines need special intervention between elections, as tabulation of data is considered a final state. Irreversibility is achieved by storing the state, encrypted and signed, in a redundant fashion, both in the hard disk of the machine and on the removable device.

When the voting software is launched for the first time, it starts in the *unconfigured* state. After the poll worker gains access to the machine (by inserting the administration smartcard and the PIN), the system allows the poll worker to enter the configuration data, such as municipality and poll site identification number. This data is necessary to determine what ballots have to be shown (e.g., multilingual, specific ballots for local polls). Once the configuration data is confirmed by the poll worker, the machine goes into the *configured* state, from which the poll worker runs the administrative tasks. These tasks include testing machine's functions, review the activity log—to check no operations have been performed on the machine and that the counter

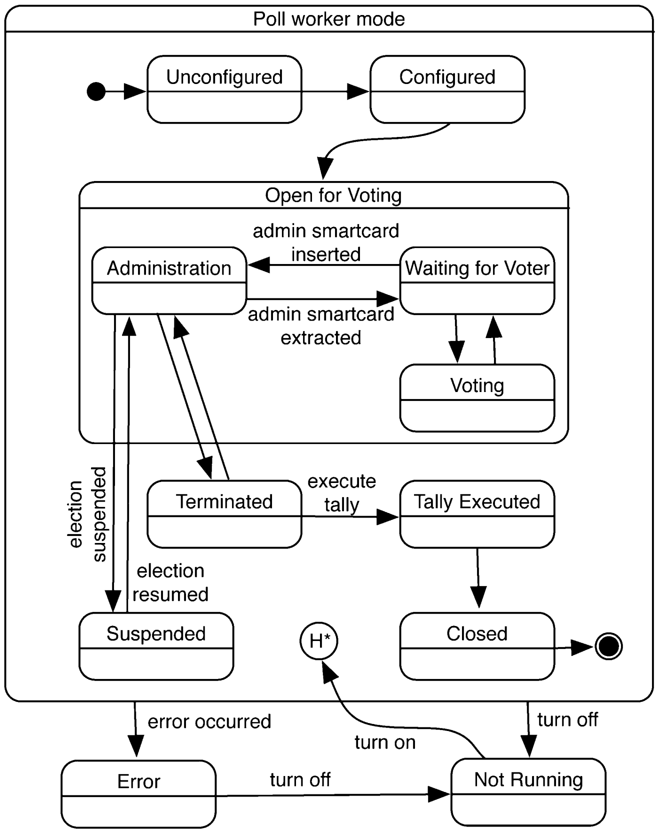

Fig. 4. Machine's poll worker mode state diagram.

of the votes is zero. From the *configured* state, the poll worker can open the election and have the machine be usable for voting.

To enable the machine for voting, the poll worker inserts one of the two *voter* smartcards in the smartcard reader. The machine can now be used for casting one vote.

The voting procedure has been designed to closely resemble that on paper, to ease transition to the usage of the electronic system. Voting activities are regulated by another state machine, shown in Fig. 5. The voter goes through a welcome screen (state *Welcome* in the figure) to a screen that reproduces the printed ballot and allows the voter to cast a choice. Once the voter confirms the choice, the machine produces a printed copy of the vote (state *Ballot Preview* in the figure) and asks the voter to verify it. The voter can now either confirm the printout, in which case the vote is stored and the machine locked, or reject it (up to two times), in which case the machine goes back to the ballot screen. The second time the only option available to the voter is making the vote void. A header (accepted, rejected, void) together with a barcode encoding the vote is printed on the ballot. The ballot is then cut and falls in the ballot box inside the machine and the machine goes into the *voted* state.

The external display shows the current status of the machine (e.g., voting, voted). When the machine goes into the *voted* state, the poll worker extracts the smartcard from the smartcard reader. To enable the machine for the next voter, the poll worker has now to use the other *voter* smartcard; this process is then repeated for all voters.

The extraction of the smartcard from the smartcard reader *before* the vote has been cast clears the machine from the pending

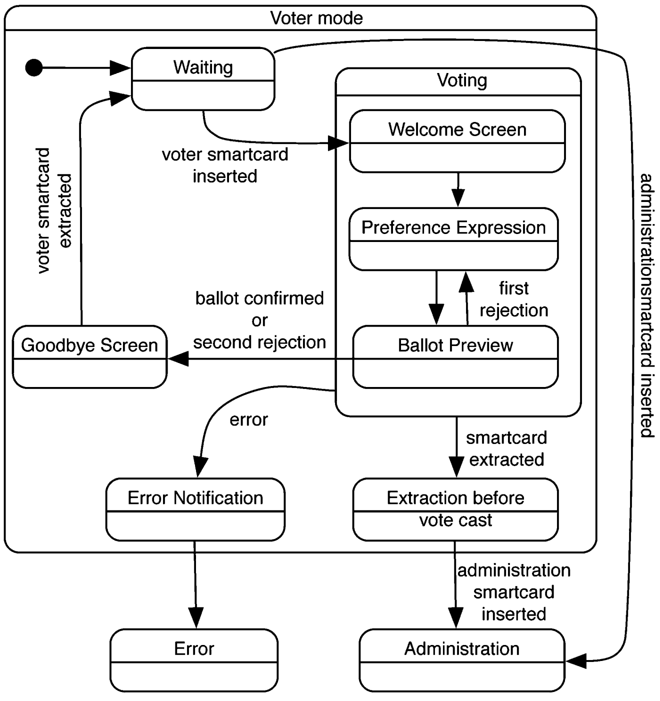

Fig. 5. Machine's voter mode state diagram.

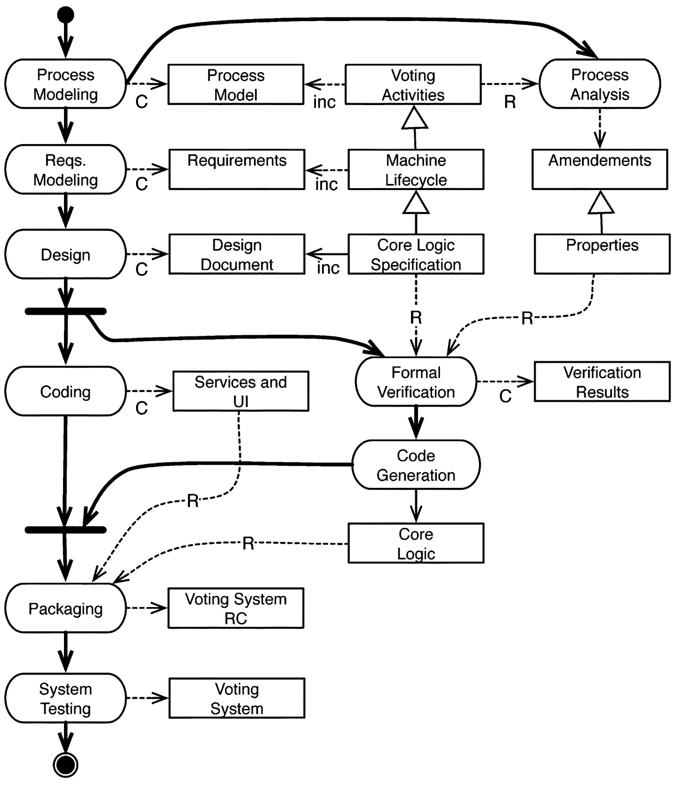

Fig. 6. ProVotE development process. The process describes the workflow that we followed while developing the ProVotE system.

voting operation (including any pending printout) and locks it. This function has been devised to interrupt a voting operation, e.g., when the voter has difficulties in using the machine, while guaranteeing secrecy. In order to avoid the risk that a voter is denied the right to cast a vote due to the extraction of the card before voting has ended, the control logic puts the machine into a state that requires special intervention by the poll worker.

During voting, the file containing the votes is stored encrypted both on the hard disk and on the flash memory. Each time a vote is cast the two copies of the file are read and compared, and the machine is blocked in case of any inconsistency. To ensure that votes are not stored in the order in which they have been cast, the set of votes is reshuffled after each vote.

D. E-Voting Machine Products

At the end of the election day, a poll worker closes the election and requires tabulation of data. Votes are counted and aggregated by the machine and the machine prints a printout of the tabulated data, together with the activity report (see below).

At this point, both the internal hard disk and the removable device contain the following data:

1) *An electronic copy of the activity report*. This is a document listing all the tasks performed by the machine including the initial counters' value, which must be zero.
2) *A file with all the ballots cast*. This is a file in XML format, digitally signed.
3) *A file containing the tabulated votes*. This is a file containing the number of votes recorded for each candidate, in XML format, digitally signed by K_p, with signature and certificates embedded in the document following the XML Signature Syntax and Processing specified in [16].

The poll workers then extract the removable devices from machines and upload files containing the results to the electoral service by means of a web application made available through a VPN. On the receiving side, the signatures of the files are verified, to ensure results have not been tampered with (e.g., by malware on the sending machine or by man-in-the-middle attacks). If the signatures verify, the data is used to determine the elected candidates. Finally, poll workers deliver the USB keys and the printed ballots to the Electoral Office, for further verification activities, if necessary.

IV. ProVotE Development Process

Broadly speaking, the development of ProVotE has proceeded in cycles, paced by experimentations and with a time-span of about six months each. Each development cycle is a waterfall [17], that we extended and adapted to incorporate (formal) verification activities. All cycles elaborated on the same set of documentation—i.e., there is just one requirement document, one design document, etc. The most interesting aspect of the development is the organization of activities within each cycle, summarized in Fig. 6.

The figure is an activity diagram whose notation we have slightly adapted to make it more readable. Activities are in rounded rectangles and artifacts in rectangles, as customary. We use a particular notation to indicate creation ("C"), reading ("R"), and inclusion ("inc") of artifacts, meaning, respectively, that an activity creates an artifact, it reads it, or that an artifact is contained in another artifact. Finally, the empty arrowhead indicates refinement, so that, for instance, the "Machine Life-cycle" refines the "Voting Activities." We omitted, in the diagram, some read operations that are standard in the waterfall model; e.g., even if not indicated, the requirement documents has been used as the basis for the design.

We distinguish, in particular, the following activities:

1) **Process Modeling**. An initial set of discussions and analysis of the Italian electoral law allowed to provide a formalization and a view of the voting procedures in Italy. The formalization was given using activity diagrams in UML and textual descriptions. Such documentation served as the basis for the subsequent requirements analysis and for the analysis of the "to be" procedures.
2) **Process Analysis**. The adoption of new technologies (be it in the polling station or in any other setting) shifts risks, threats, and attacks. If the procedures are also designed to protect against such risks and attacks (as it is for the electoral case), the problem arises of ensuring that they still make sense after the introduction of the e-voting systems. We decided, therefore, to analyze the procedures in order to understand the possible attack models and, possibly, introduce a set of additional constraints on the machines and on the laws regulating (electronic) elections.
3) **Requirements Modeling**. The initial requirements definition activity produced a requirements document, which contains high-level statecharts and use cases. The statecharts specify the life-cycle of the machine described earlier and the use cases provide the usage scenarios for each of the states of the machine's life-cycle (e.g., the actions necessary to open the poll site).
The voting and the administration user interfaces are specified through a statechart which describes their logic and in which each state corresponds to a different "screenshot" of the system (see Section III-C).
The specifications and, in particular, the UML statecharts have been discussed and validated by the Electoral Service of the Province of Trento. The voting interface was validated using throw-away prototypes developed from the statecharts, in experiments with selected voters before the trials. The experiments have been organized and set up by the Faculty of Sociology of the University of Trento.
4) **Design**. During the design phase, the statecharts validated by the Electoral Service have been detailed into *executable* statecharts.
5) **Formal Verification**. The *executable* statecharts specification have then been translated into the NuSMV input language (see below for the description, also in [18]), which, in turn, is formally verified. The translation is done automatically using FSMC+ [12], a tool we specifically developed for the purpose (see Section VI).
6) **Code Generation**. After having validated the specification of the control logic, we used FSMC+ to generate the Java code of the *control logic* of the machine (both the administration and voting part). The Java code generated by FSMC+ was inspected by hand, to mitigate risks related to bugs in the translator, and the source code then compiled and packaged to produce the voting application.
7) **Coding (and Unit testing)**. The code implementing the management of data and hardware devices and some glue code to link the user interface was then implemented and tested using standard practices.
8) **Packaging and System Testing**. The code generated by hand and that produced by FSMC+ have been finally packaged together and tested using standard techniques. Notice in fact that the formal verification performed at the previous steps does not eliminate the need for testing, as properties related to, e.g., the integration of different components, actual implementation of the drivers (is the printer really cutting the paper when the system tells it to do so?), etc. are outside the scope of the verification and, therefore, still need to be performed.

Formal verification of the machines has been based on NuSMV [18], an open source model checker. NuSMV allows for the specification of synchronous and asynchronous systems and for the verification of safety and liveness properties expressed in computation tree logic (CTL) and linear temporal logic (LTL), using both BDD-based and SAT-based techniques. Language and machinery have demonstrated to be adequate for our purposes: the control logic has been modeled as a synchronous machine and the properties expressed in CTL. The choice of the tool has also been driven by practical considerations (it is a tool for which we have extensive know-how), stability (it is used by a wide community), and availability (open source).

In the rest of the paper, we focus on two key activities of the development, namely the analysis of the processes and the formal verification of the core logic.

V. Process Modeling and Process Analysis

One of the concerns of the project is understanding how the introduction of the new e-voting system could impact the existing procedures and the verification activities that are performed to ensure no material mistakes or attempts to alter the results have taken place. (See also [19]–[21] for the importance of procedural security.) To analyze procedures in a systematic way, we defined a methodology, initially described in [13] and [15], and illustrated in Fig. 7.

The first step is the generation of what we call *extended model*. The extended model is obtained by modifying the workflows to include wrong or malicious actions, such as replacement, alteration, or deletion of an asset (e.g., a document required for the election). Thus, in the extended model, not only assets are modified according to what the procedures define, but they can also be transformed by the (random) execution of one or more threat actions.

We then transform the extended model into an executable NuSMV specification. The translation is performed by defining what we call *asset flows*, namely state machines that describe how properties of assets are changed by the execution of the extended model. A special state machine (*pc*—for process counter) encodes the order in which activities are executed. An example may help. We show a snippet of the code defining a property named `status` of the asset `electionSW` (that is, the software controlling the voting machine). The code, intuitively, states that the `electionSW` can either be `plain` or `encrypted`, that the initial state is `plain` and that the execution of the `encrypt` action changes the status of the software from `plain` to `encrypted`:

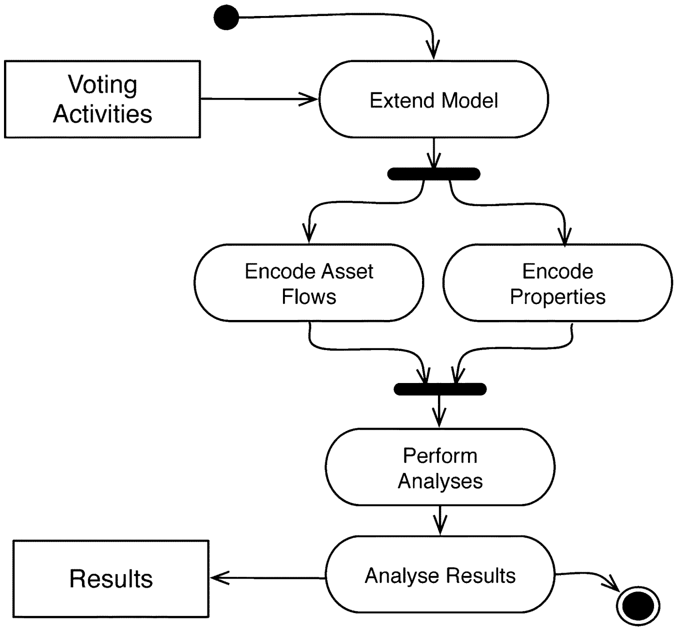

Fig. 7. Methodology for procedural security analysis.

```
MODULE electionSW (...)
VAR
  status : plain, encrypted;
  ...
  init(value) := plain;
  next(value) := case
    pc.pc = encrypt : encrypted;
    ...
```

Accessory information, such as the actors participating in each activity, can be modeled to, e.g., enrich the language that can be used to reason about the workflows. An example is the set of actors that are active at a given time. Below we show a snippet of code that defines when the actor *Electoral Service* is active. The statement uses the NuSMV `DEFINE` construct, that is used for variables that do not have an "independent" evolution:

```
DEFINE
  ElectoralServiceActive :=
    pc.pc = load ||
    ...
```

We finally model check the specification with a set of properties, expressed in LTL/CTL, that encode the security goals that have to be satisfied. Counterexamples of security properties encode the sequence of actions that have to be executed in order to carry out an attack on an asset.

We are interested, in particular, in two classes of properties (and in the counterexamples generated by such classes):

1) *Undetected attacks*, namely sequence of actions that succeed in altering one or more assets and for which the procedures provide no check to highlight the alteration. Undetected attacks are a potential menace to the democratic process: think for, e.g., undetected alteration of the electoral results in one or more polling station (see, also in [22] for more discussion about undetected attacks).
2) *Denial of services*, namely attacks which are meant to alter one or more assets in such a way that procedures have to be stopped. In the optimistic case, a denial of service in an election represents a cost and a "nuisance" for the community (as, e.g., results are delayed; the administration needs to rerun the election). In the pessimistic case, e.g., repeated attacks, it may represent a serious threat to democracy.

The following NuSMV property, for instance, shows how we can represent a *denial of service* attack. The property, expressed in CTL, states that it is never the case (`AG !`—for "Always Globally Not") that an unusable copy of the election software is delivered to the polling stations (the content of the software is "garbage" and the location of the software is the polling station)

```
AG ! (sw.content = garbageSW
     && sw.location = pollStation).
```

If the procedures are well designed, breaches to security should derive only from the execution of malicious actions.

So far we analyzed the procedures for delivering the software and the machines to the polling stations and to send the results from the polling stations back to the Electoral Office. See [13] and [15] for some more details.

VI. Formal Verification of the Control Logic

To improve our confidence in the design of the voting application, we applied formal verification techniques on a critical component, namely the logic controlling administration and voting activities. To provide an idea of the activities performed, we start by introducing the architecture of the system and we then detail the core verification activities that we have performed.

A. The Voting Application's Architecture

The *Voting Application* is a Java-based graphical application structured in four main macro-components (Fig. 8).

1) *The Services Component*. It provides the basic functionality to the rest of the application, such as drivers for controlling the external control display and the printer, managing logs for audits, and transparently managing redundant and ciphered persistence of data.
2) *The Data Model Management Component*. It manages all the election-specific data, comprising candidates and parties, the ballot data, per-machine election results, and the symmetric and asymmetric keys used for ciphering and signing.
3) *The Control Logic Component*. It defines how the machine has to react to user actions, both in the administration and in the voting mode. The control logic also specifies the logic of the user interface (e.g., what screen has to be shown next). Its architecture is shown in Fig. 9. An *Event Queue Manager* handles all asynchronous events (e.g., user actions, smartcard inserted/removed events, power fail events from the UPS) and feeds the *State Machine*, which reacts

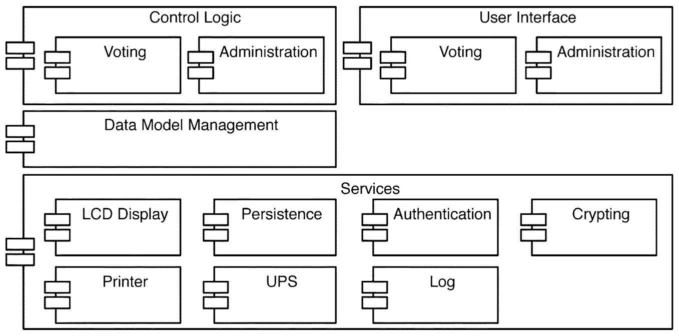

Fig. 8. ProVotE voting machine's architecture.

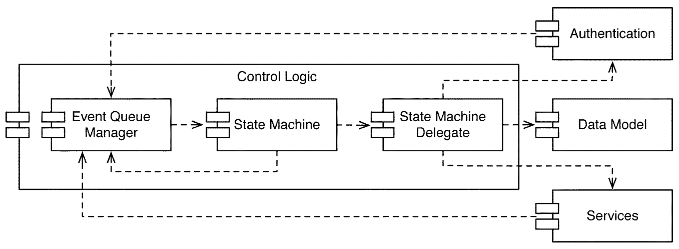

Fig. 9. Control logic detailed architecture.

to the events and sends commands to a *Delegate*. The *Delegate*'s methods can be thought like atomic building blocks (e.g., "print the current ballot," "clear the interface," "turn the outside red light on," etc.) that are combined by means of the statechart specification.

4) *The User Interface Component*. It manages the graphical layout of the administration and of the voting interfaces.

B. Formal Specification and Verification of the Core Logic

The control logic of the voting machine is among the most critical components, since any problem in its specification might lead to losses of functions or breaches to security. For this reason, we decided to use formal method techniques for its specification and verification. We did so, by building a tool called FSMC+ that allows for automatic generation of the source code of control logic from the statecharts after they are formally verified.

Starting from the detailed statecharts that describe the *control logic* component, FSMC+ automatically produces NuSMV models implementing the UML execution semantic defined in [23]. The generated model resembles the component's architecture shown in Fig. 9. Namely, it is decomposed into three submodules: *State Machine*, *Event Driver* (an abstraction for the Event Queue Manager), and *Delegate*. The *State Machine* is generated automatically while only a skeleton is produced for the *Event Driver* and the *Delegate* models.

In order to perform the verification, users have to complete by hand the skeletons of the delegate and driver modules, namely, the inputs that the machine can receive (*Event Driver*) and the actions that the machine performs (*Delegate*). In the model of these modules, for example, we specified the semantics of the actions that controls the state for hardware parts, such as the external indicator, the printer (e.g., whether a vote is exposed in the printer or not), the screen (e.g., which form is shown to the user), the internal state of memory, etc. We also modeled the behavior of the smartcard reader and of the touchscreen. Choosing the proper level of abstraction for these specifications has proven fundamental to obtain useful results.

Properties used for verification are derived from the requirements documents and from encoding a set of "sanity" checks on the specification. One can express these properties using CTL or LTL logic. The CTL properties are mainly expressed in terms of actions that the control logic has to perform on the peripherals to ensure that no basic principle of the law is violated and that the behavior of the machine is compliant with the specification provided in the requirements document.

For the sake of brevity, below we show examples of the properties specified using CTL.

Property i): There is always a way for the voter to end the voting procedure once s/he started.[4]

```
AG (dr.Stable & sm_GUICtrl.state =
     Welcome ->
     EF (dr.Stable & sm_GUICtrl.state =
     Idle))
```

Property ii): When the voter starts to use the machine, the internal representation of the vote must be empty.

```
AG ((dr.Stable & sm_GUICtrl.state =
     Welcome) ->
     (del.vote = vote_empty))
```

Property iii): During the e-voting session, if two paper ballots were printed, then the first one must show "invalidated."

```
AG (del.ballot_cast = 2
     -> del.printed_ballot_1.printer_header
     = printer_header_invalidated)
```

In this way, we have specified 41 properties corresponding to each component—i.e., the DRE, Printer (VVPAT), and External Display—of the machine as liveness and safety properties. We also categorized them into two groups those related to administration (8 properties) and voting (33 properties).

The properties along with the model of the system have been passed to the NuSMV system in order to perform the actual verification. Once the verification is performed, FSMC+ produces the Java code that we glue with the other code of the machine.

The structure of generated Java code follows the *State* design pattern defined in, e.g., [24]. According to the pattern, each state of the state machine is encoded with a Java class, that has two methods, `Entry` and `Exit`, corresponding, respectively, to the actions to perform on entering and exiting the state. Standard book-keeping machinery manages the transition to the next state. The `Entry` and `Exit` methods use the State Machine

[4]The NuSMV models obtained from UML statecharts define a boolean variable called `Stable` (e.g., `dr. Stable` for the driver module instance **`dr`**), to model the semantics of **run-to-complete** step as defined in OMG/UML [23] specifications. Some properties, in fact, have to be checked only in a stable state.

Delegate class for the concrete implementation of the actions to perform, as shown in Fig. 9.

It is worth mentioning that the use of FSMC+ and NuSMV do not guarantee that the system is error-free. This is due to, e.g., incompleteness of properties, potential bugs in the translators, potential bugs due to the integration of the actual Java components, and the implementation of the components excluded from the formal verification (e.g., OS, drivers). Standard testing activities, therefore, have been performed on the system.

VII. Related Work

The level of interest e-voting has attracted is a witness of the importance it has in the implementation of our democracy [1], [4], [9], [25]–[28].

Scientific literature on e-voting is wide and multidisciplinary. We organize previous work in four areas: understanding the risks posed by the introduction of e-voting systems in the polling stations; assessing existing systems; designing novel voting schemes, protocols, and/or techniques; and designing better e-voting systems using formal methods.

With respect to the first area, work in the past has focused on understanding what changes could be introduced in the "traditional" voting procedures to allow a secure transition to electronic elections. For instance, [29] and [30] discuss risks and difficulties related to the introduction of e-voting, [19] and [31] suggest possible improvements to existing procedures, and [13], [14], and [20] introduce techniques to formally analyze what security breaches may be derived by executing the procedures in the wrong way.

With respect to the second area, assessing existing systems, some e-voting systems currently deployed in elections have recently undergone a thorough and independent scrutiny to evaluate their quality. See, for instance, [32]–[36], where the authors highlight some serious design and implementation flaws, which could be exploited to compromise elections. The papers also suggest a drastic change in the way in which electronic voting systems are designed, developed, and tested.

With respect to the third area, we mention a number of schemes, prototypes, and/or techniques for better voting machines. Works like [37]–[40] attempt to provide (maximum) secrecy and/or anonymity for the vote and voter. End-to-end verifiability of the integrity of critical steps in the voting process and of election results is presented in [41] and [42]. What is most common to all these approaches is that they rely on the underlying cryptographic principles to various degree of complexity. Other works, such as [43] and [44], apply techniques used in other domains—like prerendering user interface and hardware separation—to build an ccessible, verifiable, and secure voting system. We should be clear that the (non-)monolithic design mentioned in our work is different from the one proposed in [43], where, from what we understand, the authors propose hardware isolation techniques to ensure voter privacy. Moreover, the choice of Java for our implementation is for the same advantages mentioned in [43].

With respect to the last area, the usage of formal methods in the specification and verification of e-voting systems is relatively new. We mention [45] and [25], which present the formal specification and verification of an e-voting protocol using Pi calculus, and [46], where the authors present a mobile implementation of an e-voting system and used formal verification technique to validate the security property of their proposed system.

VIII. Discussion and Conclusion

The development of e-voting systems is extremely challenging and demanding. The need to balance conflicting requirements, such as traceability and privacy, liveness and security, adds to the complexity of building and deploying application on which our democratic rights might depend.

In this paper, we have presented the main activities we conducted to develop the ProVotE system, an e-voting machine for the Autonomous Province of Trento. To address the issues mentioned above, we adopted a development process that tries to take into account not only aspects related to a careful development of the software, but also the environmental conditions in which the machines are actually used. We believe that such an approach is an essential cornerstone for the development of e-voting solutions we can trust as citizens.

The ProVotE machines have been used with six experimental trials by about 28 000 citizens and, in two small elections, with legal value, by about two thousand more. We shipped the software in different configurations to handle five different types of elections (with different voting and tabulation rules) and different hardware components (corresponding to different "versions" of the control display, of the screen, of the PC).

During the experimentations, we collected and analyzed data about machine performance (reboots, errors, operational errors, etc.), citizens' and poll workers' opinions. Trials are excellent testbeds. Polling stations are scattered in the territory and under the authority of people with the most diverse backgrounds, motivations, training, and capacities.

Errors, both in the execution of the procedures and when operating the machines, are made: keys to open the machine are lost, machines are delivered to the polling stations in a "blocked" state (because they have not been "cleaned" and cannot be used), under the pressure of voters queueing to vote, liveness is perceived as more important than security (e.g., machines are kept enabled for voting, because it is just faster than enabling it when a voter arrives); voters need to "adapt" to the new technology. Procedures and systems were demonstrated to be robust enough to manage all these situations.

Introduction on a large scale of the system, in any case, will require improvements to the interface, strengthening training, improving motivation and awareness of poll workers and public administrators in the usage of the new systems, and improving awareness of citizens on the real (versus the perceived) risks and advantages of e-voting.

References

[1] J. W. Bryans, B. Littlewood, P. Y. A. Ryan, and L. Strigini, "E-voting: Dependability requirements and design for dependability," in *Proc. First Int. Conf. Availability, Reliability and Security (ARES'06)*, Washington, DC, 2006, pp. 988–995, IEEE Computer Society.

[2] C. Lambrinoudakis, S. Kokolakis, M. Karyda, V. Tsoumas, D. Gritzalis, and S. Katsikas, "Electronic voting systems: Security implications of the administrative workflow," in *Proc. 14th Int. Workshop on Database and Expert Systems Applications (DEXA'03)*, Washington, DC, 2003, p. 467, IEEE Computer Society.

[3] A.-M. Oostveen and P. V. den Besselaar, "Security as belief user's perceptions on the security of e-voting systems," in *Electronic Voting in Europe*, 2004, pp. 73–82.

[4] M. Volkamer and M. McGaley, "Requirements and evaluation procedures for eVoting," in *Proc. Second Int. Conf. Availability, Reliability and Security (ARES'07)*, Washington, DC, 2007, pp. 895–902, IEEE Computer Society.

[5] S. Myagmar, A. Lee, and W. Yurcik, "Threat modeling as a basis for security requirements," in *Proc. 2005 ACM Workshop on Storage Security and Survivability (StorageSS'05)*, New York, 2005, pp. 94–102, ACM Press.

[6] Federal election commission*, 2002 Voting System Standards* [Online]. Available: http://www.eac.gov/voting

[7] F. E. Commission*, 2005 Voluntary Voting System Guidelines (VVSG)* [Online]. Available: http://www.eac.gov/voting

[8] *Legal, Operational and Technical Standards for e-Voting*, ISBN 92-871-5635-2, Council of Europe, 2004.

[9] R. T. Mercuri and L. J. Camp, "The code of elections," *Commun. ACM*, vol. 47, no. 10, pp. 52–57, 2004.

[10] L. Caporusso, C. Buzzi, G. Fele, P. Peri, and F. Sartori, "Transition to electronic voting and citizen participation," *Electronic Voting*, vol. 86, pp. 191–200, 2006, R. Kimmer, Ed.

[11] C. Buzzi, A. Brighenti, and L. Caporusso, "Translating void and null ballots from paper to touchscreen," in *Proc. Towards e-Democracy: Participation, Deliberation, Communities*, 2006.

[12] R. Tiella, A. Villafiorita, and S. Tomasi, "FSMC+: A tool for the generation of java code from statecharts," in *Proc. 5th Int. Symp. Principles and Practice of Programming in Java (PPPJ'07)*, New York, 2007, pp. 93–102, ACM.

[13] K. Weldemariam, A. Villafiorita, and A. Mattioli, , A. Alkassar and M. Volkamer, Eds., "Assessing Procedural Risks and Threats in e-Voting: Challenges and an Approach," in *VOTE-ID*, ser. Lecture Notes in Computer Science. New York: Springer, 2007, vol. 4896, pp. 38–49.

[14] K. Weldemariam and A. Villafiorita, "Formal procedural security modeling and analysis," in *Proc. 3rd Int. Conf. Risks and Security of Internet and Systems (CRiSIS'OS)*, 2008, pp. 249–254, IEEE.

[15] K. Weldemariam and A. Villafiorita, "Modeling and Analysis of Procedural Security in (e)Voting: The Trentino's Approach and Experiences," in *Proc. USENIX/Accurate Electron. Voting Technol. on USENIX/Accurate Electron. Voting Technol. Workshop (EVT'08)*, Berkeley, CA, 2008, USENIX Association.

[16] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, XML Signature Syntax and Processing (Second Edition), w3c Recommendation Jun. 2008 [Online]. Available: http://www.w3.org/TR/xmldsig-core

[17] W. W. Royce, "Managing the development of large software systems," in *Proc. IEEE WESCON*, 1970, pp. 1–9.

[18] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An open source tool for symbolic model checking," in *Proc. 14th Int. Conf. Computer-Aided Verification (CAV'02)*, London, U.K., 2002, pp. 359–364, Springer-Verlag.

[19] A. Prosser, R. Kofler, R. Krimmer, and M. K. Unger, "Security assets in e-voting," in *Electronic Voting in Europe*, 2004, pp. 171–180.

[20] A. Xenakis and A. Macintosh, "Procedural security analysis of electronic voting," in *Proc. 6th Int. Conf. Electronic Commerce (ICEC'04)*, New York, 2004, pp. 541–546, ACM Press.

[21] *Procedural Security and Social Acceptance in E-Voting*. Los Alamitos, CA: IEEE Computer Society, 2005, vol. 5.

[22] R. L. Rivest and J. P. Wack, On the Notion of "Software Independence" in Voting Systems 2006 [Online]. Available: http://vote.nist.gov/SI-in-voting.pdf

[23] O. M. Group, OMG Unified Modeling Language Specification Sep. 2001.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides*, Design Patterns*. Boston, MA: Addison-Wesley, Jan. 1995.

[25] S. Kremer and M. D. Ryan, M. Sagiv, Ed., "Analysis of an electronic voting protocol in the applied pi-calculus," in *Programming Languages and Systems—Proc. 14th Eur. Symp. Programming (ESOP'05)*, Edinburgh, U.K., Apr. 2005, vol. 3444, pp. 186–200 [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/Kremer-esop05.pdf, Springer ser. Lecture Notes in Computer Science

[26] N. K. Sastry, "Verifying Security Properties in Electronic Voting Machines" Ph.D. dissertation, EECS Department, University of California, Berkeley, May 2007 [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-61.html

[27] R. Anane, R. Freeland, and G. K. Theodoropoulos, "E-voting requirements and implementation," in *CEC/EEE. IEEE Computer Society*, 2007, pp. 382–392.

[28] M. Bishop and D. Wagner, "Risks of e-voting," *Commun. ACM*, vol. 50, no. 11, pp. 120–120, 2007.

[29] A. Xenakis and A. Macintosh, "Levels of difficulty in introducing e-voting," in *EGOV*, 2004, pp. 116–121.

[30] A. Xenakis and A. Macintosh, "G2G Collaboration to Support the Deployment of e-Voting in the UK: A Discussion Paper," in *EGOV*, ser. Lecture Notes in Computer Science. New York: Springer, 2004, pp. 240–245.

[31] M. Volkamer and R. Krimmer, "Independent audits of remote electronic voting—Developing a common criteria protection profile," in *Proc. EDEM 2007—Elektronische Demokratie in Österreich*, 2007.

[32] R. Gardner, S. Garera, and A. D. Rubin, "On the difficulty of validating voting machine software with software," in *Proc. USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop (EVT'07)*, Berkeley, CA, 2007, pp. 11–11, USENIX Association.

[33] D. W. Jones*, The Evaluation of Voting Technology*, ser. Advances in Information Security. Norwell, MA: Kluwer Academic, 2003, pp. 3–16, ch. 1.

[34] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach, "Analysis of an electronic voting system," in *IEEE Symp. Security and Privacy*, 2004, vol. 0, p. 27.

[35] D. Balzarotti, G. Banks, M. Cova, V. Felmetsger, R. Kemmerer, W. Robertson, F. Valeur, and G. Vigna, "Are your votes really counted? Testing the security of real-world electronic voting systems," in *Proc. Int. Symp. Software Testing and Analysis (ISSTA)*, 2008, pp. 237–248.

[36] N. Ansari, P. Sakarindr, E. Haghani, C. Zhang, A. K. Jain, and Y. Q. Shi, "Evaluating electronic voting systems equipped with voter-verified paper records," *IEEE Security Privacy*, vol. 6, no. 3, pp. 30–39, May/Jun. 2008.

[37] A. O. Santin, R. G. Costa, and C. A. Maziero, "A three-ballot-based secure electronic voting system," *IEEE Security Privacy*, vol. 6, no. 3, pp. 14–21, May/Jun. 2008.

[38] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *Proc. Workshop on the Theory and Application of Cryptographic Techniques (ASIACRYPT'92)*, London, U.K., 1993, pp. 244–251.

[39] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections (extended abstract)," in *Proc. Twenty-Sixth Annual ACM Symp. Theory of Computing (STOC'94)*, New York, 1994, pp. 544–553, ACM.

[40] I. Ray, I. Ray, and N. Narasimhamurthi, "An anonymous electronic voting protocol for voting over the internet," in *Proc. Third Int. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'01)*, Washington, DC, 2001, p. 188, IEEE Computer Society.

[41] Z. Xia, S. A. Schneider, J. Heather, and J. Traoré, "Analysis, improvement and simplification of prêt à voter with paillier encryption," in *Proc. Conf. Electronic Voting Technology (EVT'08)*, Berkeley, CA, 2008, pp. 1–15, USENIX Association.

[42] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman, "Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes," in *Proc. Conf. Electronic Voting Technology (EVT'08)*, Berkeley, CA, 2008, pp. 1–13, USENIX Association.

[43] N. Sastry, T. Kohno, and D. Wagner, "Designing voting machines for verification," in *Proc. 15th Conf. USENIX Security Symp. (USENIX-SS'06)*, Berkeley, CA, 2006, USENIX Association.

[44] K.-P. Yee, "Extending prerendered-interface voting software to support accessibility and other ballot features," in *Proc. USENIX Workshop on Accurate Electronic Voting Technology (EVT'07)*, Berkeley, CA, 2007, pp. 5–5, USENIX Association.

[45] S. Delaune, S. Kremer, and M. D. Ryan, Verifying Privacy-type Properties of Electronic Voting Protocols Laboratoire Spécification et Vérification, France, Research Rep. LSV-08-01, Jan. 2008, ENS Cachan.

[46] S. Campanelli, A. Falleni, F. Martinelli, M. Petrocchi, and A. Vaccarelli, "Mobile implementation and formal verification of an e-voting system," in *Proc. 2008 Third Int. Conf. Internet and Web Applications and Services (ICTW'08)*, Washington, DC, 2008, pp. 476–481, IEEE Computer Society.

Adolfo Villafiorita received the M.Sc. degree with honors from the University of Genoa, Italy, in 1993, and the Ph.D. degree from the University of Ancona, in 1997.

He is a Senior Researcher with the Center of Information Technology at Fondazione Bruno Kessler (FBK-Irst), Trento, Italy. His current interests include software and system engineering, security, formal methods, and safety analysis. He has participated and lead several industrial projects related to the development of safety critical applications in the railway, avionic, and aerospace sector. He is a member of ACM.

Komminist Weldemariam (S'09) received the M.Tech. degree from the Indian Institute of Technology Bombay. He is working toward the Ph.D. degree at the Department of Information Engineering and Computer Science, University of Trento, Italy.

He is a member of eDeomcracy Unit at the Center of Information Technology (FBK-IRST). His research interests include BPR, security, software engineering, application of formal methods, and electronic voting systems.

Roberto Tiella received the M.Sc degree in mathematics from the University of Trento, Italy, in 1993.

He is a technologist with the Center of Information Technology at the Fondazione Bruno Kessler (FBK-Irst), Trento, Italy. He moved to FBK in 2003, after an experience of nine years in development of distributed software systems in the telecommunication domain. Since 2005, he participated in the ProVotE project with contributions in the definition of e-voting systems architecture and in applications of model checking techniques to support the development of such systems.