The KOA Remote Voting System: A Summary of Work To Date

Joseph R. Kiniry, Alan E. Morkan, Dermot Cochran and Fintan Fairmichael[1],
Patrice Chalin[2], Martijn Oostdijk and Engelbert Hubbers[3]

[1] School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Ireland

[2] Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, H3G 1M8, Canada

[3] Nijmegen Institute of Information and Computing Sciences
Radboud University Nijmegen
Postbus 9010, 6500GL Nijmegen, The Netherlands

Abstract. Remote internet voting incorporates many of the core challenges of trusted global computing. In this paper, we present the Kiezen op Afstand[4] (KOA) system. KOA is a Free Software, remote voting system developed for the Dutch government in 2003/2004. In addition to being Open Source, it is also partially formally specified and verified. This paper summarises the work carried out to date on the KOA system. It charts the evolution of the system, from its initial conception by the Dutch Government, through to its current status. It also describes a roadmap of milestones towards completing its next release: a Free Software, general-purpose, formally specified and verified internet voting system, that incorporates Proof Carrying Code technology for software update and allows trustworthy voting from a mobile phone. We propose that the KOA system should be used as an *experimental platform* for research in electronic and internet voting; we are *not* saying that we have solved any of the major problems inherent in voting with computers.

1 Introduction

The Netherlands is known for its forward-thinking and progressive government, laws, and policies. Unfortunately, a government's progressiveness, particularly with respect to the adoption of new technology, is sometimes contrary to the good of its citizens.

Accordingly, in order to help avoid such a situation in the adoption of remote voting technology in the Netherlands, the Security of Systems (SoS) Group at Radboud University Nijmegen became directly involved in the evaluation and development of the KOA system in 2004.

[4] "Kiezen op Afstand" is literally translated from Dutch as "Remote Voting."

1.1 Voting Machines in the Netherlands

The introduction of such a system was not as radical a development as it might be considered elsewhere. Electronic voting machines (EVMs) were introduced without controversy in the Netherlands around 1998. They have been widely used in local and national elections ever since. The primary supplier of these machines is Nedap[5], the same supplier as in Ireland.

Part of the reason that EVMs were so readily accepted is historical. The Netherlands has used digital voting machines[6] since the 1980s. Therefore, Dutch citizens are comfortable with the idea of using technology for voting. The security and reliability issues of the new generation of machines was not a serious problem at the time of their introduction, much like their adoption by other governments in the late 1990s.

Unfortunately, many aspects of these systems have not been made public, contrary to the requests of concerned parties in the Netherlands. The internals of such systems are secret and are only exposed to evaluators. Each system must be examined, according to an unknown set of criteria, before being accepted by the Dutch parliament for use in elections. Evaluation reports compiled by the national reviewer, TNO[7], are also secret.

However, as attention has been focused the world over on EVMs, the Dutch parliament has begun to re-evaluate its approach. Changes to the current systems are likely to be mandated soon, particularly with respect to voter verifiable paper trails.

In keeping with this reassessment, the Dutch parliament decided to conduct experiments with the next natural step in the use of technology for voting: remote voting using both the internet and telephone. The main inspiration is that, nowadays, many personal transactions (e.g., banking), can be carried out from arbitrary locations, so why not voting?

Indeed, it is believed by some that a remote voting system will increase electoral participation by making the process more convenient. Currently, Dutch citizens must find time during the extended business hours (08:00 to 20:00) of a single day of the working week. Furthermore, each individual must vote in a particular location near their home, which may be far from their workplace.

However, given what we know about the unreliability and vulnerability of software and networks, do the risks inherent in the introduction of such a system outweigh such benefits?

These risks, together with the methods adopted in eliminating and minimising them in the KOA system form the basis for the rest of this paper. It is organised as follows. Section 2 presents some background information on the genesis of the KOA project. Past academic work on the system up to the end of 2005 is presented in Section 3. A security assessment of the KOA system is put forward in Section 4. Current work is discussed in Section 5. Related work is compared and contrasted in Section 6. Future work is considered in Section 7 and Section 8 concludes.

---

[5] Nedap — http://www.nedap.com/.

[6] The previous-generation systems with little-to-no software.

[7] TNO — Netherlands Organisation for Applied Scientific Research — http://www.tpd.tno.nl/tno/index.xml.

2 Kiezen op Afstand (KOA)

The genesis of KOA stemmed from a promise made by the Dutch government to parliament that they would investigate possible developments to the Dutch voting system. This promise was fulfilled in the KOA experiment by allowing expatriates to vote in the elections to the European Parliament via the internet and by telephone.

However, Dutch national election law is quite explicit about what is permitted with respect to how votes may be cast. Therefore, in order to conduct an experiment in voting over the internet, some amendments to this general law were formulated. This formed the legal foundation for the KOA project.

Apart from the general rules governing internet voting, it also included some additional rules detailing a citizen's right to vote from a different polling booth other than the one originally appointed. However, in this paper we will refer to the KOA project as if it consisted purely as an internet voting experiment.

2.1 Internet Voting in the Netherlands

The elections to the European Parliament of June 2004 allowed remote voting via the internet and telephone. It was limited to expatriates who were required to explicitly register beforehand. It was thought that such a small-scale use (thousands of voters) would provide a useful real-world test for the technology.

The main reason why it was thought that an internet-based solution was suitable is decidedly non-technical. Essentially, by significantly constraining the remote voting problem, particularly with respect to the registration and voting process itself, it was believed that a "sufficiently secure" and reliable system could be constructed. In particular, the system needed to be at least as secure and reliable as the existing remote voting system which was based upon postal ballots.

The Remote Voting Process When a citizen registers to use KOA, the voter chooses their own personal access code (a PIN). Some time later, a customised information packet is mailed to the voter. This packet contains general information about the vote itself (date, time, etc.), as well as voter-customised details that are known to only that voter. These details include information for voter authentication, including an identification code and the previously chosen access code.

Also included is a list of all candidates. Each candidate is assigned a large set of unique random numbers[8], and exactly one of those numbers is given to each voter. The set of codes per voter is determined randomly but is not unique.

To vote, a registered voter logs in to a web site with their voter code and access code. They then step through a series of simple web pages, typing in their candidate codes as appropriate for their choices. The system shows the voter the actual names and parties of the candidates in question to confirm the accuracy of the vote. When a voter is finished, a transaction code is provided. This code can later be used to check in a published list that the voter's choices were included correctly in the final tally.

[8] 1,000 codes were generated for each candidate for the elections to the European Parliament in 2004.

Communication with the voting web site is secured with SSL. All votes are stored in a doubly-encrypted fashion; each vote is encrypted by a symmetric key per voter[9] and the public key of the voting authority.

2.2 Use and GPL Release

The trial during the elections to the European Parliament in June, 2004 was restricted to roughly 16,000 eligible Dutch expatriates. Expatriates could vote either via the internet or by telephone. The telephone votes were fed into the KOA tally system. 5,351 people used one or other system.

Subsequently, in July 2004, the Dutch Government released the majority of the source code for the KOA system under the GNU General Public License (GPL) making it the first Open Source internet voting system in the world.

3 Academic Past Work

3.1 External Security Evaluation

In late 2003 Prof. Bart Jacobs of the Security of Systems (SoS) group at Radboud University Nijmegen participated in an external review of the requirements and design of this application. One of the recommendations made by the panel was that the system should not be designed, implemented and tested all by the same company.

The system itself was designed and implemented by LogicaCMG[10]. Although eventually the government decided to make the system open source, during implementation it was not. In order to improve its quality, the Dutch company Software Improvement Group[11] performed a code review of the system. However, they were only allowed to do so after signing Non-Disclosure Agreements (NDAs). In fact, it was unexpected that the government ultimately opted for an Open Source solution.

The SoS group did not take part in the design or implementation of the system. However, the group took an active part in the final stages of the project. The group performed two tasks: it wrote an independent tally application which will be explained in detail in Section 3.2 and it performed a penetration test on the vote servers.

The penetration test was set up as a black box test. In particular the SoS group had virtually no knowledge regarding the actual hardware, software, networks or personnel involved with the server system. Indeed, the information it did possess could have been considered public information since it could easily be obtained by standard available analysis tools.

The main goal was to break into the system and try to compromise its integrity. The second goal was to test whether the system was vulnerable to denial of service attacks.

Two evaluations were conducted. The first was unsolicited and took place during a private beta test of the system. The second was requested by the government, primarily because of the results of the first evaluation.

[9] This symmetric key is generated by hashing the assigned identification code.

[10] http://www.logicacmg.com/.

[11] http://www.sig.nl/.

During the first unsolicited evaluation the subnet running the service was gently probed and mapped using nmap, a more detailed evaluation of specific machines was then conducted, specifically with regards to machines running inappropriate services, weaker operating systems, etc., and finally, on the last afternoon of the test, a denial-of-service attack on the machines was conducted.

The main discovery of the first evaluation was that the system was not "tightened down" insofar as test and management machines which were running insecure versions of particular operating systems (e.g., Microsoft Windows) were on the deployed subnet, no hardware or software firewall was in place on the system, machines has likely external exploits available, and nearly all systems had inappropriate services running (e.g., unused mail servers, databases, file sharing, etc.). Also, the SoS group was able to significantly harm their service quality with our (admittedly very small) denial-of-service attack.

After the authorities realized the SoS group was responsible for this attack they asked us for a report of our findings. Given the feedback and analysis, they then asked the SoS group to perform an "official" external evaluation once they incorporated all of our suggestions and tightened-down the network.

The second evaluation found that their systems were adequately hosted, monitored and configured, their software was up-to-date, and no unnecessary services were running. Furthermore, adequate measures were in place for detecting basic probes by adversaries. Thus, in the end, the SoS group did not find any problem with the system that would have caused the Dutch Ministry to reject it for an experimental run, and the external evaluation significantly improved the security and reliability of the system.

3.2 Vote Counting System

As seen in the previous section, one of the results of the recommendation to split the responsibilities of the parties involved, was that the government decided to accept bids for the creation of a separate vote counting subsystem, to be implemented in isolation by a third party. This separate tally application would allow the vote counting to be independently verified. The SoS group put forward a proposal to write this application, and were successful in this bid. The key idea behind their tender was that the vote counting program should be formally verified using the JML [2] and ESC/Java2[12] [10] tools.

The vote counting system formed a small but important part of the whole KOA system. This provided the SoS group with a suitable opportunity to test the use of some of the formal techniques and practices that they had been developing. Given the severe time constraints placed upon them due to the impending election, the application was built by three members of the group over a barely-sufficient period of four weeks. Java was chosen as the programming language in which to implement the system so that JML could be used as the formal specification language. Due to the time constraints, verification was only attempted with the core modules.

[12] ESC/Java2 is a programming tool that attempts to partially verify JML annotated Java programs by static analysis of the program code and its formal annotations. It translates the specifications into verification conditions that are modularly discharged by an automatic theorem prover.

Counting votes within KOA proceeds offline using a separate tally application. The input to this application consists of two XML files (one containing the list of candidates and their codes, and one containing the encrypted votes), and a public/private keypair used to decrypt the votes.

As the informal requirements of vote-counting are obvious (for every candidate in the candidate list count the number of votes for that candidate), the functional specification [12] (in Dutch) mostly prescribes details of file formats and encryption algorithms to be used.

Nevertheless, the functional specification does impose some requirements that greatly influence the structure of the Java application and its JML specification. First, the different tasks that need to be performed in order to count the votes (reading in the two files, reading in the keys, decrypting the contents of the votes file, counting the votes, generating reports) are made explicit in this document and, more importantly, the order in which they have to be performed is specified. Second, the document provides a rough sketch of the user interface and its contents. Finally, the document gives some bounds on the data, such as the lengths of fields or the maximum number of candidates in each list, which are incorporated in the JML specifications of the data structures.

In accordance with the above high-level specification, the resulting tally application consists of some 30 classes, which can be grouped into three categories: the data structures, the user interface, and the tasks.

The data structure classes form an excellent opportunity to write JML specifications. Typical concepts from the domain of voting, such as candidate, district and municipality can be modeled with detailed JML specifications. An example invariant in Candidate.java is:

```
/*@ invariant my_gender == MALE ||
  @           my_gender == FEMALE ||
  @           my_gender == UNKNOWN;
 */
```

The different tasks associated with counting votes were mapped to individual classes. After successful completion of a task, the application state is changed. A task can only be started if the application is in an appropriate state. The life-cycle model of the application that therefore emerges is maintained in the main class of the application inside a simple integral field. This life-cycle model can be specified in JML using invariants and constraints, essentially stating that on successful completion of the application, the application went from “initial state” to “votes counted state”. The state of attributes associated with the individual tasks can be linked to the application life-cycle state using invariants. For instance, such an invariant could read: ‘after the application reaches the “keys imported state”, the private key field is no longer null’. This is stated in `MenuPanel.java` as follows:

```
/*@ invariant
  @   (state >= PRIVATE_KEY_IMPORTED_STATE
  @     ==> privateKey != null);
 */
```

A graphical user interface is usually not very amenable to formal specification. Nonetheless, some light-weight specifications were written. One of the requirements defined in the original informal specification was that users should not be allowed to start certain tasks before certain other tasks are successfully completed. For instance, a user should (by means of the user interface) not be able to start decrypting votes before the votes are read in from file. In the graphical user interface, this demand is met by only enabling certain buttons when the application reaches certain states in the life-cycle model. The fact that the graphical user interface complies with the life-cycle model can be neatly specified in the GUI classes by referring to the application state.

3.3 Process

As already stated, ESC/Java2 was only used to verify the core of the tally application. This means that it was used to verify reading in the XML-files with the candidates and the votes, decryption of the votes and counting the votes. The final generation of the reports is not checked with JML.

Using JML on reading XML files is quite straightforward. Essentially, for every object that is read, some methods are called that specify that the total number of objects will be increased by exactly one. Naturally, in order to verify code that uses functionality provided in external libraries, some of the corresponding APIs must also be specified. The JML community has provided specifications for most of the APIs that come with Sun's standard edition of Java. However, APIs dealing with cryptography, XML parsing, and PDF generation, as used by the tally application had not previously been specified. These APIs were specified in a light-weight manner: the specifications mostly deal with purity and non-null references in the API methods which makes verification of client code using ESC/Java2 much easier.

Naturally, the counting process is likewise formally specified in JML, which ensures that each valid vote is counted for exactly one candidate. This also implies that specifications are easy to check to make sure that the total number of votes a party list receives is equal to the sum of votes for each candidate[13] on this party list.

The JML run-time assertion checker was also used in the development process. First, for testing the data structure classes, the checker was used to generate unit tests. Second, we ran the full application, including user interface, using the checker.

3.4 Analysis of KOA

In the Dependable Software Research Group at Concordia University, the KOA source code was used as a subject of a study in the frequency of occurrences of non-null reference type declarations [3]. This work consisted in adding nullity annotations (or constraints) and then verifying their correctness by making use of ESC/Java2. The results were similar to those of Fähndrich and Leino [13], that is to say, it was found that even a simple specification exercise of adding nullity annotations can help uncover non-trivial bugs both in the code and in the specifications.

[13] Including the 'blanco' or 'blank ballot' candidate.

For example, in the `sos.koa.CounterAdapter` class in the Tally Application it was found that the field named `errors` is declared nullable and yet the method `getErrors`, which uses this field, assumes that the field is non-null (i.e., a `NullPointerException` will not be thrown).

3.5 Reverse Engineering Missing Components

The version of KOA released under the GPL was not complete. A number of pieces of functionality, constituting roughly 10% of the deployed KOA system, were proprietary and owned by LogicaCMG. Moreover, certain other changes were made for publication purposes (e.g., the length of public/private key pairs in the source code).

In addition, the released KOA system contains no high-level design documentation and very little information on how to build the system. This means that it is only possible to inspect the (partial) source, not to compile and run it. Therefore, it was necessary to perform a full analysis of the released system [14].

One of the most beneficial aspects of this analysis was that errors were found in the KOA system. One such error was found in the Java Server Pages (JSPs) whereby a button that should have guided the user back to the interface homepage had, in fact, the same action as that of the "submit" button that processed and saved a list of candidates to the database. This was due to a trivial mistake: placing the HTML tags for the "Return Home" button within the `FORM` tag block. This error was discovered during a trivial "click through" of the user interface followed by an examination of the code.

Such a basic mistake in the design of the user interface of a critical system is unacceptable. The fact that such a mistake could be made, remain unnoticed in the testing and evaluation phase of the software, and actually be used in the elections to the European Parliament, would suggest that there is in all likelihood further errors in this software.

Once the analysis was complete, the missing functionality was reverse engineered. 59 additional classes, together with some properties files, were added to the system. These classes carry out the base functionality of the servlets, error reporting, logging functionality, event handling, etc.

3.6 Full Open Source Foundations

One of the major goals in the redevelopment of the KOA system was that it would be entirely composed of, and dependent upon, Open Source software. The original system was developed in, deployed upon and tightly coupled to the IBM WebSphere IDE. During the reimplementation, the KOA system was ported to an Open Source alternative. This foundation consisted of a MySQL database server backend paired with a JBoss application server front-end, the latter of which incorporated the Tomcat servlet container. The other major restriction in terms of making the system fully GPL-compliant was its use of proprietary security and encryption utilities developed by IAIK and Sun. These were seamlessly replaced using the BouncyCastle Open Source alternatives.

3.7 Formal Specification and Extended Static Checking Review

As has already been stated, the Vote Counting Application of the KOA system was specified with formal methods, extensively tested and partially verified to the extent that was possible within the given time-frame. Subsequently, efforts were made to complete the specification and verification [8].

When the KOA vote counting system was being designed, precedence was given to verifying the core units. These were designed by contract and as result have good specification coverage. The remaining parts, however, were only lightly annotated with JML notation.

	File I/O	**Graphical I/O**	**Core**
Classes	8	13	6
Methods	154	200	83
NCSS	837	1599	395
Specs	446	172	529
Specs:NCSS	1:2	1:10	5:4

Table 1. KOA initial release system summary

Table 1 summarizes the size (in number of classes and methods), complexity (non-comment size of source (NCSS)), and specification coverage of the three subsystems, as measured with the JavaNCSS tool version 20.40 during the week of 24 May, 2004. This is the version of the program that was released and used in the elections to the European Parliament in June 2004.

At the time of its initial release, verification coverage of the core subsystem was good, but not 100%. Approximately 10% of the core methods (8 methods) were unverified due to issues with ESC/Java's Simplify theorem prover (i.e., either the prover did not terminate or terminated abnormally). Another 31% of the core methods (26 methods) had postconditions that could not be verified, typically due to completeness issues in ESC/Java, and 12% of the methods (10 methods) failed to verify due to invariant issues, most of which are due to suspected inconsistencies in the specifications of the core Java class libraries or JML model classes. The remaining 47% (39 methods) of the core verified completely. Since 100% verification coverage was not possible in the time-frame of the original project, to ensure the KOA application was of the highest quality level possible, a large number unit tests were generated[14] for all core classes with the jmlunit [4] tool, which is part of the JML suite. A total of nearly 8,000 unit tests were generated, focusing on key values of the various datatypes and their dependent base types. These tests cover 100% of the core code and are 100% successful.

[14] The tool generates unit tests that deal with *interesting* values. Interesting values are generally boundary values for a given data type. For example, -1, 0, 1, n and $n+1$ for an array of integers. Users are also free to handwrite their own test cases, in the case where the jmlunit tool does not test all important values.

After this analysis was completed, the specifications were gradually augmented. As an example, consider the `AuditLog` class. This class records information about the vote counting as the application proceeds. This information is then used at the end of the vote counting to help fill in the details for two of the reports that are generated. This class keeps track of the program's progress in a similar manner to that which was used for the overall program state. There were multiple invariants used to ensure the program and auditing proceeded in the correct fashion. Several corrections were required for this class, the bulk of which were modifications to the behaviours of the methods that allowed the audit log's state to change. The original specifications allowed the possibility that the variables could be changed to a state where the invariants would not hold. The changes made to this class' specifications disallowed any actions that would violate the object invariants.

3.8 Documentation Writing and Translation

The vast majority of the voting system, including high-level documentation, web interfaces, Java comments and variable names are in Dutch. Furthermore, much of the voting system is sparsely commented and unspecified. This clearly poses an obstacle to the understanding and adoption of such a system by a wider, international audience. It was therefore decided at an early stage that a complete translation of the system into an international language such as English, together with the production of additional documentation, was necessary in order to facilitate a larger number of people to carry out the necessary specification, development and testing. Consequently, the major high-level specification document and all of the JSPs have been translated from Dutch into English.

3.9 Other Voting Systems

Naturally, there are relatively considerable variations in electoral systems between countries. This is the case between the Netherlands and Ireland. Not only are these differences linguistic, but more significantly there are different vote counting procedures in the Netherlands and in Ireland. The Dutch Voting system is list based while Ireland uses Proportional Representation with a Single Transferable Vote (PR-STV).

The Irish Voting System The Dáil, Ireland's lower house of parliament, is composed of 166 members representing 41 constituencies. Each constituency elects multiple members to parliament. The average constituency elects four representatives with every constituency electing at least three representatives. The system used is PR-STV. This combination is considered to increase the representativeness of the Dáil.

Irish voters, by ranking the candidates, give instructions as to who should receive their support should the first choice candidate be eliminated or elected. Surplus votes are the number of votes in excess of the threshold of election a candidate receives. Surplus votes are transferred proportionally to the remaining candidates according to the indicated second preference of the voters. If the election is undecided after counting the first preferences and transferring surplus votes, then the lowest polling candidate

is eliminated. The ballots cast initially in support of this candidate are now counted according to their indicated second preference. If any candidate has more than a quota of votes then he or she is elected and his or her surplus votes are transferred to the next preference candidate. If there are more candidates than seats and all surpluses have been transfered, then the candidate with least votes is excluded and his or her votes transfered to the next preference on each ballot paper. This process is repeated until the number of candidates remaining equals the number of seats remaining.

Formal Specification Votáil is the Irish word for voting. The Votáil specification is a JML specification for the Irish vote counting system [5]. This formal specification is derived from the complete functional specification for the Dáil election count algorithm [6,7].

Thirty nine formal assertions were identified in the Commentary on Count Rules published by the Irish Department of Environment and Local Government. Each assertion expressed in JML was identified by a Javadoc comment. In addition, a state machine was specified so as to link all of the assertions together. Java classes were specified for the vote counting algorithm, to represent the ballot papers and candidates. A concrete example of how the methodology was applied will clarify this work.

Section 7, item 3.2 on page 25 of [6] states:

> As a first step, a transfer factor is calculated, viz. the number of votes in the surplus is divided by the total number of transferable votes in the last set of votes. This transfer factor is multiplied in turn by the total number of votes in each sub-set of next available preferences for continuing candidates (note that the transfer factor is not applied to the sub-set of non-transferable votes in the set of votes).

The requirement is translated into formal natural language as follows:

> The number of votes in the surplus is divided by the total number of transferable votes in the last set of votes. This transfer factor is multiplied in turn by the total number of votes in each sub-set of next available preferences for continuing candidates.

Finally, this formal natural language is formally specified in the architecture as a JML postcondition for the method that is specifically for this requirement (the `getActualTransfers` method). The Javadoc and JML specification for this method follows.

```
/**
 * Determine actual number of votes to transfer to
 * this candidate, excluding rounding up of
 * fractional transfers
 *
 * @see requirement 25 from section 7 item 3.2
 * on page 25
 *
 * @design The votes in a surplus are transfered in
```

```
 * proportion to the number of transfers available
 * throughout the candidates ballot stack.  The
 * calculations are made using integer values
 * because there is no concept of fractional votes
 * or fractional transfer of votes, in the existing
 * manual counting system. If not all transferable
 * votes are accounted for the highest remainders
 * for each continuing candidate need to be examined.
 *
 * @param fromCandidate Candidate from which to
 *        count the transfers
 * @param toCandidate Continuing candidate eligible
 *        to receive votes
 * @return Number of votes to be transfered,
 *         excluding fractional transfers
 */

//@ ensures
//@  \result ==
//@     (getSurplus(fromCandidate) *
//@      getPotentialTransfers(fromCandidate,
//@         toCandidate.getCandidateID()) /
//@      getTotalTransferableVotes(fromCandidate);
```

The Votáil specification was typechecked and checked for consistency using ESC/-Java2[15].

4 Security Assessment

Issues of security and correctness are paramount in any voting system. This is especially the case for a remote internet voting system due to the inherent vulnerabilities of the architecture. Any such system must be as secure as the system it is designed to replace. Otherwise, trust in the electoral and democratic systems of a country can be severely damaged.

The KOA system was designed to replace absentee postal ballots. It has always been accepted that postal voting is not as secure as voting in a polling booth. KOA follows all of the standard security mechanisms and also introduces some novel approaches. These security mechanisms are focused on attack prevention and, where this is impossible, on detection of intrusion. This section discusses these security mechanisms.

4.1 Data Integrity

The most significant method used in the KOA system to ensure data integrity is the use of candidate codes. 1,000 codes are generated for each candidate and only one of

[15] The consistency of JML specifications is checked using an experimental extension to ESC/Java2 that manipulates the JML abstract syntax tree in order to determine whether certain combinations of assertions are inherently unsatisfiable.

these is randomly assigned to each voter. Therefore, even if a malicious agent (e.g., a worm, virus or Trojan horse) can access a ballot, all the attacker can see are the encoded candidate and party IDs, which in the optimal case are unique to the voter in question. Consequently, it will be virtually impossible to substitute the ballot by choosing the appropriate code for a different candidate.

In addition, the votes are doubly-encrypted. The only way to decrypt these votes on the server side is to close the polls. Closing the polls is an irreversible action. Consequently, altering the votes at the server-side is precluded.

In the case where the voter tries to cast multiple votes at once (e.g., via both telephone and internet) there will always be one first vote. This vote will be stored. The second attempt will fail because the voter has already cast his/her vote.

Finally, the KOA system has the capability to take snapshots of the candidate and voter lists called "electronic fingerprints." These fingerprints can be generated at any time to ensure that these lists have not been maliciously altered. One possible extension to the system is to automate the generation of these fingerprints at regular intervals to ensure a regular verification of data integrity.

4.2 Verifiability

Voters using the KOA system are able to verify that their vote is recorded correctly and is included in the final tally of the election using the transaction code they receive upon casting their ballot. This is possible due to the publication of a list of the transaction codes of votes for each candidate after the election. Such a check can identify any compromised PCs and in the worst case invalidate the election.

4.3 Insider Threats

The power to change the state of the system and to decrypt the votes is restricted to a small number of polling station officials. These officials hold the private key for the system and each has a PIN code to use this private key. One of these officials is designated as the current "president" or "chairman."

In order to change the state of the system (e.g., open/close the polls, decrypt the votes, etc.), the chairman and one other official must enter their PIN codes. If the role of chairman is alternated at set time intervals among random officials (or some similar mechanism), then all officials need to be in collusion in order to tamper with the system. Even then, access to the decrypted ballots is precluded, as is mentioned in Section 4.1.

In addition, an insider attack would require massive, undetectable client and/or network subversion (e.g., large numbers of client computers *and/or* network web proxies being compromised by a virus written by attacker's henchmen). Given the scale and complexity of such an attack, it is nearly inconceivable that it is possible. Such an attack would be (many) orders of magnitude more difficult to pull off than any attack on existing electronic or manual voting hardware/mechanisms due to its scale: millions of PCs versus thousands of voting machines, and millions of individuals (many of which are experts like network service providers, IT workers voting from home, etc.) participating and monitoring the election versus thousands of volunteers running the election.

This is analogous to the Open Source "thousands of eyeballs" argument, but applied to voting.

4.4 Other Security Features

A part from the use of SSL, there are a couple of further noteworthy security features.

Firstly, random data is added to the votes when they are encrypted. This ensures that votes within the same voter district and for the same candidate have a different encryption result for each vote, making it impossible to interpret encrypted votes.

Secondly, the votes are decrypted in a random order in order to making tracing voters by the order in which they voted impossible.

4.5 Problems

Despite the best efforts to make KOA as secure as possible, certain security flaws still remain. These need to be addressed before further use of the system.

Firstly, if the electronic fingerprints of the system are not identical at a particular point in time, the chairman can overrule and allow the election to continue. This should not be permitted.

Like other forms of remote voting (e.g., postal voting), KOA does not provide protection for voter anonymity in the case where another person is in the vicinity of the voter during the voting process or if another person gains access to a voter's transaction code. However, due to the use of candidate codes, excluding these two scenarios, it is virtually impossible to connect a voter to his/her vote.

Denial of Service Attacks (DoS) As has already been stated in Section 3.1, the KOA system is vulnerable to DoS Attacks. This is practically impossible to prevent and is a feature of all remote internet voting systems.

One feature of the KOA system that lessens some of the problems caused by DoS attacks is that the system can be interrupted. When this state change happens, an electronic fingerprint of all the system data is taken and this can be checked against a subsequent fingerprint on system resumption. Clearly, this does not solve the problem of potential temporary disenfranchisement, but it does ensure data integrity in the face of a such an attack.

4.6 Summary

As has been described, all of the standard security mechanisms have been used together with some innovative techniques to ensure data integrity and verifiability. However, obviously the issue of security is one of the open questions of remote internet voting and there are a number of problems yet to be overcome. We believe these problems can be addressed by research and experimentation on a verified open source framework, like the one which KOA aims to provide.

5 Academic Current Work

5.1 Generalisation of System for non-Dutch Voting Systems

The Java code for Votáil was written in JML using a kind of "verification-centric" Design by Contract methodology. This means that not only are we writing each method implementation according to its JML specification, but we are checking each method's correctness with ESC/Java2 and automatically generating thousands of unit tests using JML-JUnit [4].

The KOA system has a state machine similar to that used in the Votáil specification. This allows KOA to make calls to the appropriate part of the Votáil code. The `ElectionAlgorithm` class in Votáil will be invoked from within the KOA system using the following four method calls: `setup`, which defines election parameters such as candidate list and number of seats, `load`, which loads all valid ballots and then calculate quota and deposit saving thresholds, `count`, which assign votes to candidates, distribute surpluses and exclude candidates until finished, and `report`, which reports the election results. These methods must be called in the order shown, and this fact is captured by the invariants of the state machine. Only the `report` method is called more than once for each instance of the `ElectionAlgorithm` class.

The user interface is being designed in a flexible fashion so as to present non-Dutch ballot papers to the voter. The original KOA system was designed for use with a party-list system with a single national constituency. Its user interface is being extended in line with the guidelines for the Irish voting system. The KOA system allows the voter to select a list of candidates. In the Irish system each candidate is in a list of size one. The KOA system allows only one selection by the voter. In the Irish system the voter makes multiple selections in order of preference.

6 Related Work

6.1 A Security Analysis of SERVE

The security analysis of the SERVE project [9] is one of the best known examinations of remote internet voting. It is very critical of current efforts and advises against any use of such methods given the current state of technology, due to its inherent vulnerabilities.

Two main arguments against internet voting can be distinguished in the report. Firstly, it is argued that the system allows for vote buying and selling. However, this holds for any voting system in which voters vote at home. Internet voting can only be fairly compared to postal ballots, not to voting at polling stations. If we want to introduce remote voting on a large scale, measures can be taken (technical, organisational, and legal) that make it unattractive to buy or sell votes.

A second argument against internet voting is that the technology is vulnerable to attacks. Unfortunately, despite claiming to have examined alternatives to the SERVE system, it ignores systems that have overcome some, but not all, of the problems mentioned. Although, the KOA system was not fully developed at the time of writing, the recommendations presented in 2002 by Dr. Rolf Oppliger[16] for the use of a remote in-

[16] *How to Address the Secure Platform Problem for Remote Internet Voting in Geneva* — available from http://www.ifi.unizh.ch/~oppliger/Docs/sis_2002.pdf.

ternet voting system in Geneva[17], describe security mechanisms, such as code sheets, that the authors of the SERVE report do not mention.

KOA is a much more secure system than SERVE in that it uses code lists for data integrity, transaction codes for verifiability and is not closed and proprietary.

6.2 The RIES System

The RIES system was developed for elections for public water management authorities in the Netherlands. It has two main features which create confidence in the limited possibilities of attacking the system. First of all, a reference table is published before the elections, including (anonymously) for each voter the hashes of all possible votes, linking those to the candidates. It is possible to compare the number of voters in this table with the number of registered voters. After the elections, a document with all received votes is published. This allows for two important verifications:

1. A voter can verify his/her own vote, including the correspondence to the chosen candidate.
2. Anyone can do an independent calculation of the result of the elections, based on this document and the reference table published before the elections.

If your vote has been registered incorrectly, or not at all, it can be detected. And if the result is incorrect given the received votes, this can also be detected. The main technique that achieves this is the clever use of hash functions. Whereas the hashes of all possible votes are public, it is impossible to deduce valid votes from them without the required voter key. Of course, the relation between voter and voter key should not be stored anywhere, as is the case for bank access codes. The system has worked well in an actual election with 70,000 voters.

A disadvantage of the RIES system in comparison with the KOA system is that a voter needs to compute hash values in order to verify that a vote has been correctly recorded. This is far more complicated than simply checking a transaction code in the list of votes after the election.

7 Future Work

Several pieces of future work have been identified and some of them are currently underway by researchers at UCD.

7.1 Development of a Mobile E-Voting Application

The EU MOBIUS Project[18], of which UCD and Nijmegen are both members, focuses on several topics including the specification and verification of security properties at several levels.

[17] http://www.geneve.ch/evoting/english/welcome.asp.
[18] The MOBIUS Project — http://mobius.inria.fr/.

As part of this work, the security properties, including a functional specification, for a MIDP-based remote voting application are in the process of being defined. An example of such a security property is: "The application must not have access to personal information (e.g., phone book) on the mobile phone".

Additionally, a MIDP-based remote voting applet has been developed at UCD. This application has been reviewed and will be refactored, including the security and functional requirements expressed in JML, for incorporation into KOA.

7.2 Full-blown Verification

We intend to fully specify and verify critical subsystems of the KOA system as a case study for the new MOBIUS Integrated Verification Environment (IVE) that is being developed by UCD and others. This goal is much more ambitious than simply performing extended static checking on various critical classes.

7.3 Just-in-Time Deployment with PCC

One of the primary problems with electronic voting systems is that new software updates, at both operating system and application levels, are typically installed in the field without any certification [11]. One technology that can help solve this deployment issue is Proof-Carrying Code (PCC) [1,15], the primary underlying formal foundation and technology used by the MOBIUS IVE.

Using a PCC technology foundation, new system and application patches could be just-in-time deployed to the thousands of voting machines used in an election with complete assurance. Developing such a foundation is part of the MOBIUS project's mandate, so the KOA system may be used as a deployment case study in the coming years.

7.4 American Voting System

The American voting system is the focus of an intense amount of discussion and work, given the ongoing fiasco in electronic voting we have witnessed in the U.S. over the past several years.

After integrating the Votáil Irish voting subsystem, we would be interested in collaborating to formally specify and verify a voting subsystem for use in American presidential and/or congressional elections using the same verification-centric methodology we have followed thus far.

7.5 Electronic Voting Systems

An electoral-system independent, formally specified and verified remote voting system can be used in an electronic voting system, as the latter is just a trivial, non-remote version of the former. It is our intention to build and demonstrate such a system, incorporating a new formally specified and verified voter-verifiable paper trail subsystem.

7.6 Reflections Future Plans

Many of these plans are "just" a matter of good software engineering and thus can be accomplished by undergraduate and postgraduate students as case studies, theses work, etc. Others are *much* more difficult. In particular, attempting verification in any form and incorporating PCC techniques into the system are quite difficult, time consuming, and even require new research to be conducted. This work will take several years to accomplish, and only if the number of individuals and groups working on and with the system grows over time.

8 Conclusion

The availability of an American voting subsystem will make KOA the first general-purpose, formally specified and verified remote and local voting system available in the world, and furthermore it will be available under the GPL license. Furthermore, the KOA system is being donated to the UK Grand Challenge Verified Code Repository as a major case study for the application of formal methods to critical, large-scale software development.

It is unclear how to compare such a system to the current commercial and Free/-Libre/Open Source Software (FLOSS) voting systems being proposed by others, given that none of them, to our knowledge, even write formal specifications, let alone perform verification. We hope that this work will encourage other similar projects to seriously consider the use of lightweight formal methods in such critical systems development.

While integrating the Votáil subsystem into the KOA system, and prior to/during the new full FLOSS foundation release of KOA, a number of new pieces of English documentation and functional specification must be written. Given that remote voting is a key case study in verified computing, we hope that the availability of such documentation and specification will provide additional motivation for researchers and developers to seriously consider using the KOA system as a foundation for Verified Verifiable Voting (VVV).

We propose that the KOA system should be used as an *experimental platform* for research in electronic and internet voting; we are *not* saying that we have solved any of the major problems inherent in voting with computers. We encourage researchers interested in electronic and internet voting to contact us and join this effort.

9 Acknowledgments

This work is being supported by the European Project Mobius within the frame of IST 6th Framework, national grants from the Science Foundation Ireland and Enterprise Ireland and by the Irish Research Council for Science, Engineering and Technology. This paper reflects only the authors' views and the Community is not liable for any use that may be made of the information contained therein.

References

1. Elvira Albert, Puri Arenas, and Germán Puebla. An Incremental Approach to Abstraction-Carrying Code. In *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, Lecture Notes in Computer Science. Springer–Verlag, November 2006.
2. Lilian Burdy, Yoonsik Cheon, David Cok, Michael Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. An Overview of JML Tools and Applications. *International Journal on Software Tools for Technology Transfer*, February 2005.
3. Patrice Chalin and Frédéric Rioux. Non-null References by Default in the Java Modeling Language. In *Proceedings of the Workshop on the Specification and Verification of Component-Based Systems (SAVCBS 2005)*, September 2005.
4. Yoonsik Cheon and Gary T. Leavens. A Simple and Practical Approach to Unit Testing: The JML and JUnit Way. In Boris Magnusson, editor, *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP 2002)*, volume 2374 of *Lecture Notes in Computer Science*, pages 231–255. Springer–Verlag, June 2002.
5. Dermot Cochran. Secure Internet Voting in Ireland using the Open Source Kiezen op Afstand (KOA) Remote Voting System. Master's thesis, University College Dublin, March 2006.
6. Department of Environment and Local Government, Commission on Electronic Voting. Count requirements and commentary on count rules, June 2000.
7. Department of Environment and Local Government, Commission on Electronic Voting. Count requirements and commentary on count rules, update no. 7: Available surpluses and candidates with zero votes, April 2002.
8. Fintan Fairmichael. Full Verification of the KOA Tally System. Final Year Undergraduate Project Thesis, March 2005.
9. David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. Analyzing Internet Voting Security. *Communication of the ACM*, 47(10):59–64, 2004.
10. Joseph R. Kiniry and David R. Cok. ESC/Java2: Uniting ESC/Java and JML: Progress and issues in building and using ESC/Java2 and a report on a case study involving the use of ESC/Java2 to verify portions of an Internet voting tally system. In *Construction and Analysis of Safe, Secure and Interoperable Smart Devices: International Workshop, CASSIS 2004*, volume 3362 of *Lecture Notes in Computer Science*. Springer–Verlag, January 2005.
11. Jason Kitcat. Source availability and e-voting: an advocate recants. *Communications of the ACM*, 47(10):65–67, 2004.
12. LogicaCMG. Kiezen op Afstand: Hertellen Stemmen. Functional specifications, 2004.
13. M. Fähndrich and K. Rustan M. Leino. Declaring and Checking Non-Null Types in an Object-Oriented Language. In *Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA 2003)*, pages 302–312, New York, NY, USA, 2003. ACM Press.
14. Alan E. Morkan. KOA Evaluation, Demonstration Installation and Implementation. Final Year Undergraduate Project Thesis, March 2005.
15. George C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1997)*, pages 106–119, New York, NY, USA, 1997. ACM Press.