IT-UNIVERSITETET I KØBENHAVN

Front page for written work/project

Course Hand-in

Course at ITU study board	Course at EBUSS study board
☐	☐ Synopsis ☐ Mini Project ☐ Other written work

Project (with Project Agreement)

ITU-Main supervisor	EBUSS-Main supervisor
☐ **Thesis** ☐ **Finishing Project** ☐ **Project**	☐ **Thesis** ☐ **Project** ☐ Summer Project ☐ 4-weeks Project ☐ 12-weeks Project ☐ 16-weeks Project

Course, project or thesis title:

_____

Course manager/Supervisor(s):

_____

_____

Name(s):	Birthdate and year:	ITU-mail:
1._____	_____	_____@
2._____	_____	_____@
3._____	_____	_____@
4._____	_____	_____@
5._____	_____	_____@
6._____	_____	_____@
7._____	_____	_____@

Courses with e-portfolio:
Link to e-portfolio: _____

Table of contents

Abstract

In the past few years a lot of countries around the world have been developing their own systems of electronic voting. E-voting offers a great opportunity to increase participation of the voters, helps to make fast vote counting and delivery of the final results, reduce the cost of holding elections in the long term. However, even if these systems bring some theoretical advantages, they are not immune (like any other systems) to security flaws that were not detected during the development. Protection of voting systems from the possibility of hacking and manipulating the results is a critical issue as it threatens the adherence to democracy principles.

Norway conducted their first digital voting in their recent national elections in September 2011. The aim of this work is to evaluate security of the new e-voting system by performing static analysis of the code of the system using advanced security tools and techniques. During the work a number of security problems were found and analyzed, and possible solutions proposed.

Introduction

The development of information and communication technologies largely determines social and political development of each country. One promising application of modern information technology is the development of electronic voting (E-Voting) systems.
The term of electronic voting means the application of technologies for vote production and counting, and tabulation of results by any electronic tools.
There are two main types of e-Voting systems

- Electronic voting at the polling station with the use of electronic technologies, such as optical scanning system that automatically reads from the paper ballot, or a system of direct recording via the touch screen or push-button terminal.
- Remote voting with the use of Internet or other communications (phone, mobile phone).

Attempts to introduce electronic voting began long time ago. First it was United States of America who invented first type of e-voting machines at polling places and used them for the presidential election in 1964. After that in a late 90's and in the beginning of 2000's a number of other countries started to use different types of e-Voting machines at their polling stations. They were Australia, Belgium, France, United Kingdom, Portugal, Brazil, Ireland, Germany, Netherlands, India. A little later some countries as Australia, Canada, Estonia, France, Spain, Switzerland, UK, USA started to introduce remote voting systems [2].
Most of these countries are continuing to use e-voting in elections at various levels (from university elections to parliamentary) and plan to develop it further. Last few years their numbers have been augmented by some other countries. Despite the fact that more and more countries are looking towards implementing remote voting systems, the popularity and trust to electronic polling station machines among voters in different countries is significantly bigger. Mistrust is even present when using e-voting machines at polling stations. For instance, Germany, Ireland, The Netherlands has stopped their e-voting project because of a number of publications about weakness of Nedap e-voting machines used at elections in these countries [2]. For remote e-voting systems situation is more complicated. In this case the voter is not supervised and its computer is not controlled by electoral authorities as the voting machines located at polling stations. This fact leads voters into questioning the security of these systems even more. Nevertheless, there are plenty benefits of using the remote voting system. First of all, it helps to make fast vote counting and delivery of the final results. Moreover, it does not depend from location of the voter and helps involve people to the electoral process, who can not take a direct part in the elections because of various reasons (for example physical disabilities, residence abroad, etc.). The society, however, still meets many obstacles on the way to introduction of these systems. World's experience shows that the implementation and effective use of electronic voting requires a long and concerted effort of the state, political and social organizations, and experts to be able to overcome the technical, organizational and legal problems that will appear on the way of implementing this system.
Today there are only a few countries where e-voting systems became successful and widely used: India and Brazil (polling place e-voting machines), and Estonia (Internet voting). However, a lot of countries are still trying to develop and introduce their own e-voting systems for wide use. Among them is Norway, who already has used polling place machines for almost 10 years [2] but for the election in September 2011 it was decided to develop a new Internet voting

system [3]. The building verifiability of this kind of system is the first priority. Norway's open information policy has been used to facilitate the accomplishment of this priority. They provided a possibility to learn the system by third parties. The big amount of logical and technical documentation and source code were shared with public for testing, as well as for reviewing and evaluation of the system and particularly its reliability.

Norwegian E-voting system

The project was started in August 2008 by Ministry of Local Government and Regional development. Its purpose was to "establish a secure electronic voting solution for general, municipal and county council elections which shall provide better accessibility for all user groups to cast their ballot. The solution shall ensure rapid implementation of elections with efficient resource usage in the municipalities, and shall facilitate the exercise of direct democracy. The present high level of trustworthiness at holding elections, based on the principle of secret ballots, shall be upheld." [3] In September 2011 voters in 10 municipalities in Norway had the opportunity to cast their vote electronically in the municipal and county elections.
The system was developed by EDB ErgoGroup, a Norwegian information technology company, in partnership with Scytl, a Spanish company specialized in Internet voting solutions [4]. According to the information obtained from Christian Bull, who is an employee of the Ministry and one of the lead developers of the eVoting project, the Ministry also participated in system development.
By taking the advantage of Norwegian open information policy, the documentation related to the system and the source code was taken into consideration, which further was studied and evaluated in depth. The E-vote solution's functionality was acquired through the documentation and code shared on website of Ministry of Local Government and Regional Development.
To explore the functionality of the code first of all the documentation regarding the system has been studied. The Ministry provided a lot of information on its website regarding the subject: video presentations of the project, legal framework documents relevant for e-voting, evaluation reports of the trial and technical documentation. The technical documentation included detailed description of system requirements, use cases, overview of the system architecture, security target and architecture of all three subsystems and operational environment. All this documentation provided a better understanding of the system's purpose and functionality, however it did not give a deep description of system implementation. There were insufficient information regarding the actual implementation: documentation regarding the code, technologies used, class (flow, sequence) diagrams. So it was difficult to study the system functionality in detail. However, the information acquired was enough to get the needed knowledge.
The source code was checked out from Ministry subversion server. Right now the two versions of code are published: the initial release imported on 03.06.2011 and version 1.0.7 imported on 06.10.2011. The latter is the version used on the elections and evaluated in this thesis.
The implementation of E-voting solution contains three different systems:

- Election administration system written in Java (~80%) and Perl (~20%) (95,796 lines of code),
- Electronic counting system written fully in C# (36,244 lines of code),
- and Electronic Voting system written in Java (184,806 lines of code).

Each system operates in its own environment, but has to interact with other two systems to satisfy common requirements for configuration, authentication, reporting, auditing and key management. [6] One part of Administrative system is called Electoral roll (ER). It holds all the eligible voters' data. Prior to the election voters can request to be added to the ER through the eVote system. Just before election starts, the eVote system receives a copy of electoral roll to generate return codes (for voter's verification of the correctness of the ballot that he cast). The parties can submit their list of candidates through a Party list system before the election starts. When election starts, the eVote and eCounting systems receive a zip file with election configuration from Administrative system including candidate list. During the election the eVote system queries the Administrative system to verify that a voter is in ER. The Administrative system receives daily updates to ER from Norwegian Revenue Service (SKD) and sends statistics (counts) to Central Bureau of Statistics (SSB). The eVoting system is implemented in a such way that each voter can cast his e-vote unlimited number of times to overwrite his vote cast previously. Moreover, p-vote overwrites the e-vote, but p-vote cannot be overwritten by e-vote: the p-vote has higher priority. To allow that, the election officials export electoral roll with mark offs of submitted p-votes from Administrative system and send it to eVote system to eliminate voters that have voted on paper. This is done in several iterations during the election day to allow reporting of preliminary result to Administrative system. The preliminary results are also reported by eCounting subsystem. After the election day both systems: eVote and eCounting provide Administrative system with final results. The eVote system performs cleansing (deletes all ineligible votes) before the submission of final result to Administrative system.

The entire communication of Administrative system with municipal networks, election sites, SSB, SKD, eVoting and pVoting components goes through secure session. All certificates are distributed as part of the installation process. The certificates for system components that are residing outside the data centers are generated by responsible system administrators and distributed by removable media via mail [6].

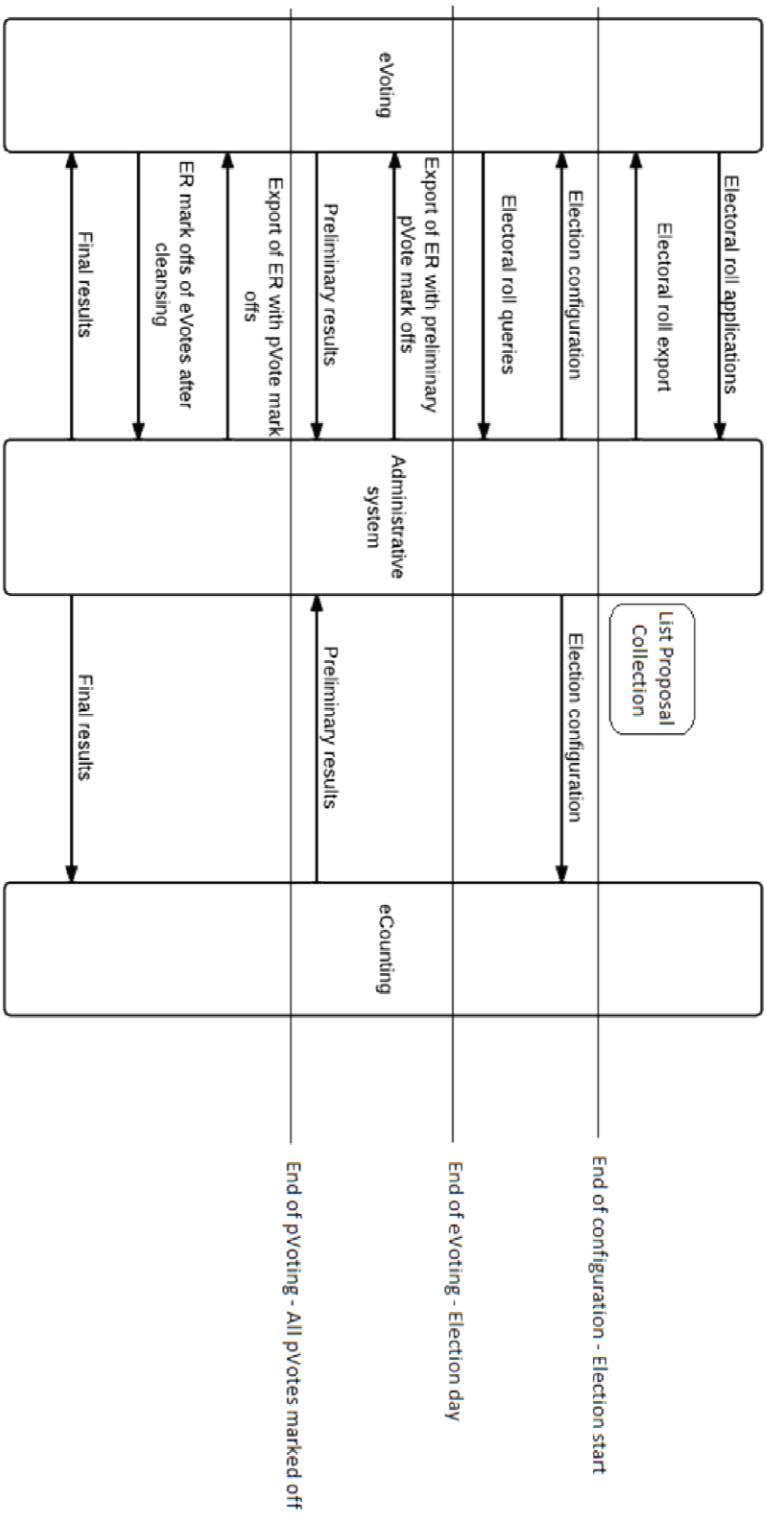

Figure 1. Communication between systems. [6]

All the systems: eVote, eCounting, Administrative and Party list system have the front and back end. The Administrative system consists of the electoral roll, election configuration, reporting, module for approving candidates lists, counting module to accept vote counts from eVote and eCounting systems, module to perform settlement of an election, two supporting modules for managing role base access and scanning of candidates lists and to external interfaces to SKD and SSB. The eCounting system is installed in municipal counting centres and contains modules for ballots scanning and verifying, and local administering. The eVote system consists of its own administration module, modules needed for a voter to submit his vote and receive a return code for vote verification, and module that performs cleansing, mixing and counting operations on the submitted votes. Additionally there is a separate key management service that supports secure communication between different parts of the system. Each system generates its local log files that are continuously sent to the central logging system. Moreover, all the systems have external interfaces to ID-porten, the national provider of electronic IDs that are used for authentication [6].

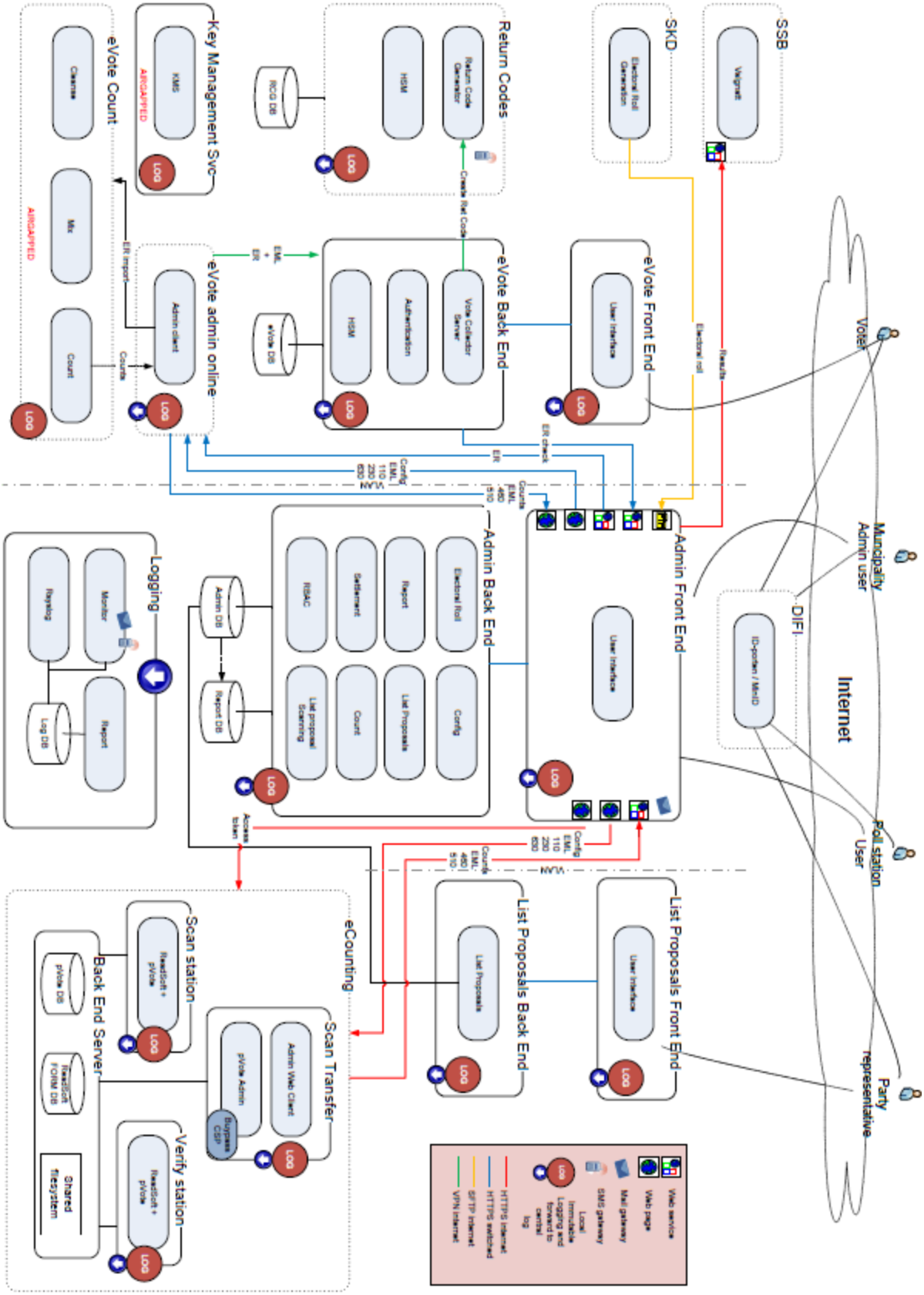

Figure 2. Overall architecture. [6]

The main focus in this thesis is on eVote system's solution. The eVote system includes client and server implementation, and protocol for electronic voting. The implementation of client application is a Java applet, and the Java Servlet Technology was used to extend Web server capabilities. The project was developed using features of the Spring Framework and built with Apache Maven.

The lifecycle of voter's interaction with the eVote system is following. First of all, the voter downloads and runs the client applications. After successful authentication the Authentication service sends voter credentials including private key for signing a vote to the voter's computer. Then the system checks voter against the Electoral Roll and presents valid elections/referendums to the voter based on their rights in the ER. The voter can select the preferred party and candidates. After the vote is cast, the system makes a preliminary eVote mark off against the voter in the Electoral Roll. Then the voter receives the return code to his mobile phone. This code is used to check if the vote submitted by voter is the same that was received by the server. Prior to the election the voter receives a mail with the paper voting card that contains all party and candidate codes that were personally generated for him. After the casting of vote and receiving the SMS with return codes the voter can verify if his vote was recorded correctly. To do that he matches return codes with the codes from voting card that corresponds to his party and candidate choice [7].

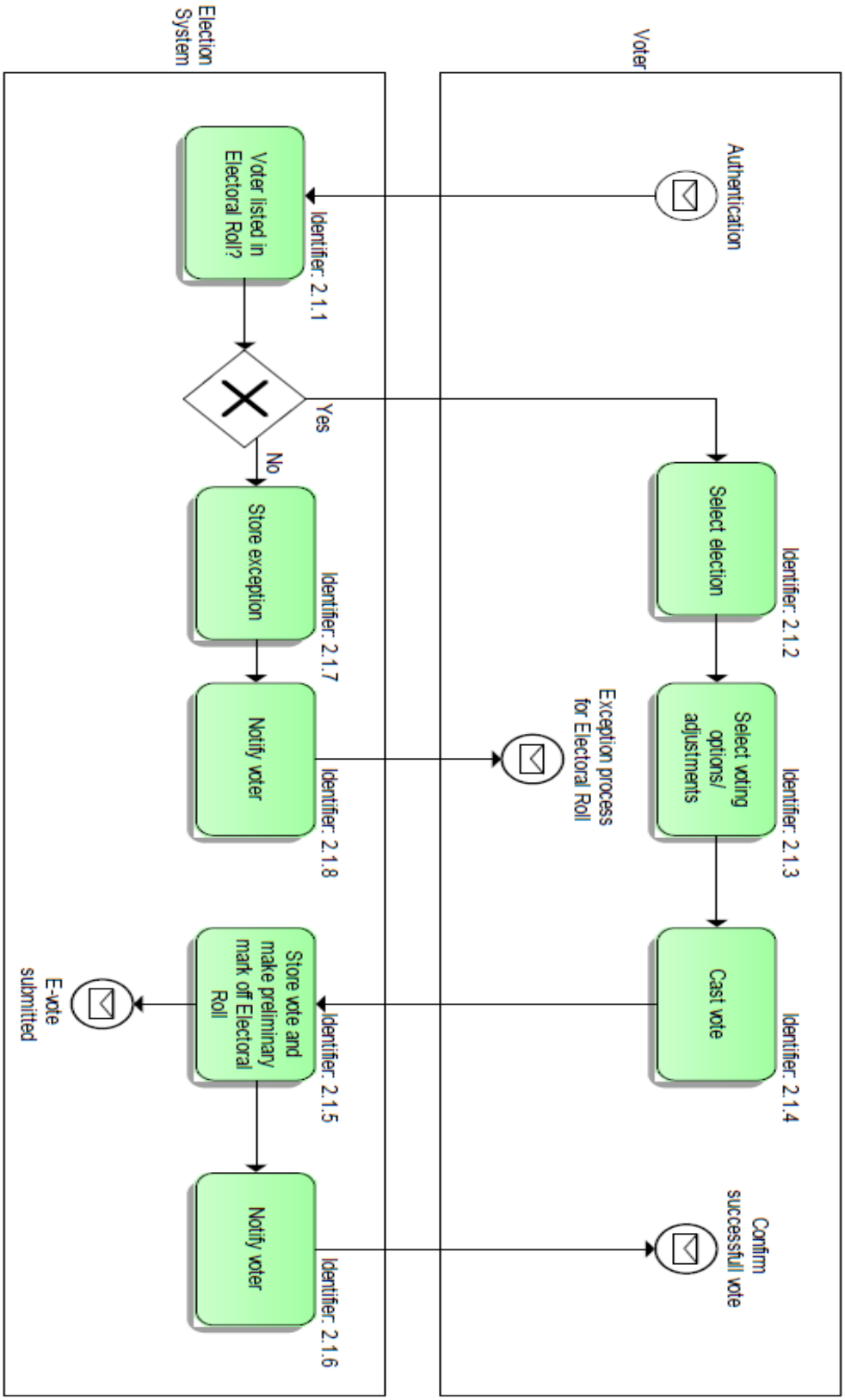

Figure 3. E-Vote. Main Flow [7]

The amount of operation performed by this system is not limited by the procedure described above. As it was mentioned earlier, the functionality of eVote system includes, for example, votes' cleansing, mixing and counting operations, and sending and receiving sensitive information such as p-vote marks off, election configuration. However, the procedure of casting a vote is a key process and that's why is more interesting from the security point-of-view. The subsequent work was mostly focused on eVoting procedure implementation.

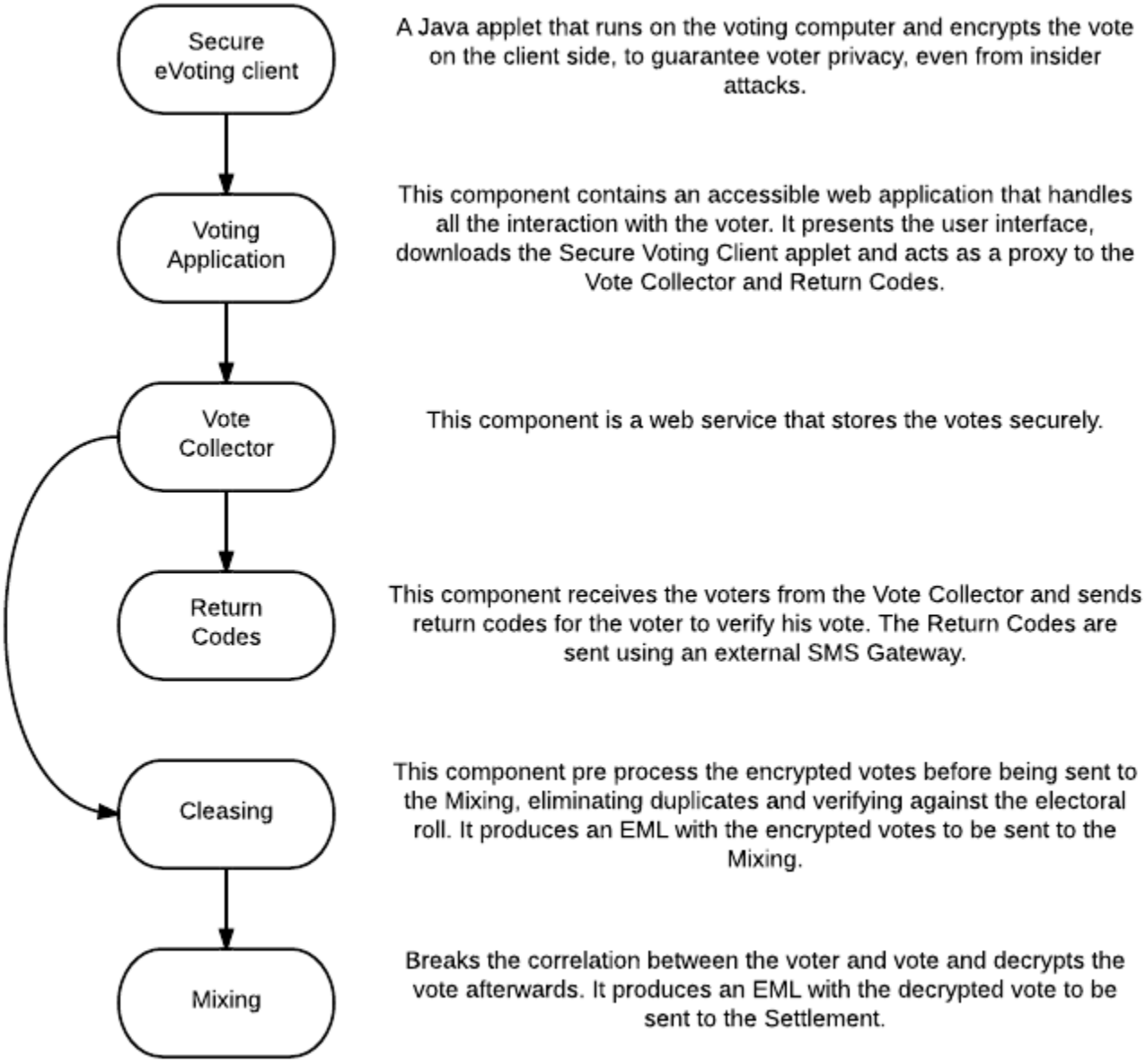

Figure 4. eVote data flow through the components of eVoting domain [5].

Problem formulation

In systems like e-voting systems reliability plays a very important role, because the user passes a lot of sensitive data through the application. The first goal of the developers of Norwegian E-voting system was to achieve high level of security and privacy of the voter [5]. Having this in mind the developers were looking for a development platform that has been designed with built-in security capabilities. The common opinion is that compared with other platforms Java is more reliable. Often by reading a literature regarding Java, such as Java documentation, specification, articles, research papers, the claims that Java was developed with security in mind are observed [8], [9], [10]. Another reason of choosing this platform is that the application written in Java can be run on all of the most popular platforms. The advantage of such program execution method is its full independence from the operating system and hardware. The assumptions is that it was a reason why the developers of Norwegian E-voting system have made their decision in favor of Java platform.

As it was mentioned previously, the front end application of the eVote system is a Java applet. The Java applet is a type of mobile code. Such kind of application should be downloaded and run on user's computer. This is usually an automatic process and does not require user's direct interaction. The mobile code is a big issue in security world. Nowadays, code is downloaded from the Internet and executed transparently by millions of users and can hide all sorts of hazardous code. Two major components are falling under concept of secure execution of mobile code application. The first is how secure is the work performed by the code; the second is the container that executes it [1]. The developers of Java platform did their best to develop secure environment for mobile Java code execution.

The Java-based applications can be executed on any platform to which the Java Virtual Machine (JVM) has been ported. The JVM itself is an instance of the JRE (Java Runtime Environment) that has the set of standard class libraries (reusable code like util, math, lang, awt, swing etc). This virtual machine starts on the execution of Java program. After the execution is complete, the instance is removed by the garbage collector. The main parts of the JVM are the class loader, the bytecode verifier, the linker, and the bytecode interpreter. The combination of classloader, bytecode verifier, and security manager refers to the term "sandbox" that is used to create a secure environment for Java applications to run and prevent malicious applets from accessing the resources on a client machine.

The execution of Java program is done in the following order. Firstly, Java source code (that may consist of a set of classes placed in different .java files) is translated into byte code (stored as .class files) by Java compiler (javac). The class loader loads each class file when it is first needed during application runtime, reads it and arranges the bytecode instructions in a form suitable for the verifier. Then the bytecode verifier performs checks on the incoming bytecode (both for local compiled source code and applet). Only after that it can be executed by the virtual machine. SecurityManager class checks applications permission to perform potentially dangerous operations based on the level of trust to this code during runtime. The fully untrusted code has only minimum of abilities: ability to access the client's CPU, memory to build objects and to the Web server from which the applet was downloaded. The responsibilities of bytecode verifier is making sure that class files contain only legal Java bytecodes and that they behave in a consistent way (no attempts to overflow the stack, no illegal pointers to memory, etc).

Together with the Java virtual machine it guarantees language type safety at Runtime. A class loader loads the classes and defines namespaces to ensures that one applet cannot affect the rest of the runtime environment. Finally, the linker resolves interfile references, and the bytecode interpreter translates it and executes machine-specific instructions.
Each of the three components of sandbox must work properly in order for Java to perform in a secure way. All in all, the three parts strongly connected to create a default sandbox. The verifier protects both the ClassLoaders and the SecurityManager against language-based attacks meant to break the JVM. The class loader system is protected by the SecurityManager, which ensures that an applet cannot create and use its own ClassLoader. ClassLoaders protects in its turn the SecurityManager from spoofing attacks by protecting local trusted classes making up the Java API. The SecurityManager also depends on ClassLoaders to correctly label code with the level of trust and help it to apply proper restrictions on the applets. [11] [12]
Even though Java seems to be secure, there are still ways to overcome its security defences and break it. No system (language, application) can be developed 100% secure and Java is not an exception. The optimal way to deal with this is minimizing the risks. The first version of Java was released in 1995. Already in a few months after that, researchers have found several serious flaws. Some of them allowed full system penetration (manipulating with private data stored on user's computer, injection of viruses etc.). During the year 1996 8 different Java flaws were discovered, including, for example, the ways to overcome the rule that an applet can connect only to the server from which it was downloaded from or how to execute malicious code as completely trusted local code.
One of flaws found during that time were connected with a bug in dynamic-linking process. A hostile applet was able to define a subclass of the ClassLoader which was actually not allowed. One of the tasks of bytecode verifier is to check that constructor of subclass always call the constructor of its superclass or another constructor of the same class. The original ClassLoader was implemented in a such way that if the ClassLoader is constructed by an applet it tells to SecurityManager to generate a Security Exception. However, researchers have found a way to overcome this rule and construct their own ClassLoader. The fix of this bug was the following: Sun Microsystem decided to split a defineClass method, which is responsible for returning a class instance, into two methods: private and public. The private defineClass method did the actual work of returning the instance of the class requested, the public method checked if the original ClassLoader was called. If not then it refused to return the requested class instance. It seemed that bug was fixed. However, just a couple of months later researches have found a way to perform the same kind of attack using another Java flaw: a rare case in which Java fails to check whether a method is private. Using this flaw they could execute private defineClass method directly.
The last security flaw in that year was found in August, this was followed by a quite long break when no more flaws were found and those that were found were fixed. Some Java votaries have even begun to claim that Java does not have bugs anymore and now it is completely secure. However, already in February 1997 with the release of JDK 1.1 two new and serious security flaw was found and during year 1997 it was followed by another 5 flaws in the row. The reason why it happened is obvious: all of the biggest flaws (or most of them) of the first version were discovered, but the new functionality brought new ones. [11]

Whole this “Java life story” proves once again that none, even the most secure system is not completely secure and developers can not fully rely on it. One more thing to keep in mind is that Java's cross-platform capability is not advantage only for users and developers but also for attackers: their malicious code can be executed almost on any platform.
Of course it is not a new technology, the developers are fixing and improving it for years, so the assumption is that the quality is much better than it was in its early releases, however, it is not certain that it is ideal. Each new release is followed by a number of updates, which often include security bug fixes. The last version of Java was released this year in July. Already in August the big security flaw was found. An attacker could use social engineering techniques to entice a user to visit a link to a website hosting a malicious applet. Using a vulnerability found in Java SecurityManager a Java applet could grant itself permission to execute arbitrary code. [13]
Even though the question of what flaws Java implementation has is significant, it is not the only one to be concerned about. As in the beginning of this section was mentioned, another crucial question is how developers make flaws in their systems written in Java. Software developers programming in Java also have security problems to handle.These problems are not about how secure language implementation is, but whether the programmers use it properly, or do they know about most common ways to exploit a system on the language level. A hacker for breaking a system can use different ways: breaking the code of the system, components of JVM or class libraries that the application relies on. Following best practices, for example, such as Secure Coding Guidelines for the Java Programming Language published on Oracle Corporation official website or “Guidelines for Java Developers listed in Securing Java. Getting Down to Business with Mobile Code” book [11] written by Gary McGraw and Ed Felten could help to minimize the risks.

Evaluation methods

The code review is an important part of development of secure system. One part of it is manual reviews by developers themselves and persons that were not involved in development process. The second is automated code analysis, which also should be part of the code review procedure because it can make it significantly more efficient. For this purpose a number of different tools are used. These tools by following certain rules or bug patterns try to find pieces of code that can be incorrect, inefficient, insecure, poor style or otherwise suboptimal. There are two types of automatic code analysis tools: dynamic (with code being executed) and static (without code execution). As different code analysis tools have different approach to code analysis and follow different (but likely overlapping) set of rules they can have different understanding of the code they are studying. They produce reports for developer’s use in determining the quality of the system under construction and as a guide for planning its improvement .
The goal of this work was to perform a static code analysis of the Norwegian e-voting system. The first step of this work was collecting and studying literature regarding the system and subject. At the same time the automatic static code analysis of the whole system was performed. For this purpose the following automatic static code analysis tools were used.

- PMD is a static ruleset based Java code analyzer that scans source code and looks for potential problems, possible bugs, unused, duplicate or suboptimal code, over-complicated expressions. PMD comes with a set of rules. As PMD works on source code, the problems that are reported by it are not necessarily errors, but rather inefficient code problems. These problems could be violation of naming conventions, lack of curly braces, misplaced null check, long parameter list, unnecessary constructor, missing break in switch, Cyclomatic complexity, etc. Even if do not corrected these problems, the application could still function properly. However, it can find the problems that badly affect systems well-functioning. The rules that I had focus on are called Security Code Guidelines. [17]
- FindBugs is an analysis tool, which operates on a compiled product. FindBugs works on bytecode. Here are some problems FindBugs finds which PMD doesn't: equals() method fails on subtypes, clone method may return null, reference comparison of Boolean values, impossible cast, 32bit int shifted by an amount not in the range of 0-31, a collection which contains itself, equals method always returns true, an infinite loop, etc. Each of the problem FindBugs can find corresponds to one of FindBugs bug pattern - a code idiom that is often an error, all the bug patterns are divided into a category. The bugs that I focused on belong to Security category. [18]

The reason why these two tools were chosen is that both of them are the most popular open source Java code analyzers and complement each other. There is a good amount of overlap between them, but each provides a unique service and finds a different set of problems so as much aspects of good coding practice as possible could be covered. Together they can help to maintain a high level of code quality across the project. These tools were used to scan the whole byte and source code of the eVote system for potential problems.

The main criteria for selecting packages to be evaluated are based on the tools' reports and information presented in the system documentation. The statistical data that have been conducted from the reports was used in order to detect the package that has the biggest amount of warnings. The reports of two tools were strongly overlapping. In summary 221 different potential bugs were found: in auditing package were found 15 bug in 6 classes, authentication - 10 bugs (+3 found in spring framework used in this package) in 5 classes (+2 classes of spring), counting - 23 bugs in 10 classes, evoting - 16 bugs in 6 classes, protocol - 47 bugs in 22 classes, reporting - 24 bugs in 4 classes, vsframework - 86 bugs in 31 classes. The majority of them were connected with encapsulation violation problem. Others were an improper use of arrays as immutable objects, package protection and static field being public but not final. The last two were later considered as false positives. The biggest amount of bugs were reported in protocol, reporting and vsframework packages, and the biggest amount of warnings per line of code - in reporting, auditing and vsframework. The biggest amount of classes affected were in vsframework and protocol packages. According this statistical computations, the protocol, reporting and vsframework packages were chosen for further consideration.

The secondary criterion was the importance of the functionality of package evaluated and the significance of the data it exposes. All the bugs found in protocol and vsframework packages were connected with encapsulation violation problem. The reporting package has more varieties of warnings: two of them were connected with improper use of mutable arrays. However, other warning about encapsulation violation found in reporting package was of less interest comparing

to two others packages: exposing values of dates comparing to exposing of such data as signatures, vote's and ballot's data, etc. If to compare reports for vsframework and protocol package, the amount of sensitive data exposed in the code in protocol package seemed bigger. So, it was decided to choose for detail consideration the bugs found in protocol package. Also, since all the flaws found in the protocol package belong to encapsulation violation problems, it is interesting to consider some other type of security problems. For this reason it also was decided to analyze two bugs about improper use of mutable arrays from reporting package.
As a consequence, the protocol package functionality was chosen for detailed security evaluation based on the PMD and FindBugs reports, as well as reporting package functionality for partial evaluation. The bugs found in both packages were analyzed with more focus on their influence on eVoting procedure well-functioning. Moreover, the possible solution to the problems detected was presented in this thesis.
The other five packages were not considered in this work, though there were also found a number of potential problems. Perhaps, if the whole system has been analyzed in details, it would be found out that the code that were not taken for evaluation is more weak and prone to attacks. This code also operates with some amount of sensitive data that has been exposed at some point. For example, the code from auditing package exposes references on objects that hold passwords, the code from counting packages exposes objects with signatures, etc. Also, as it was previously indicated, the vsframework package has big amount of warning about exposing of sensitive information. However, the volume of this work does not allow a detailed assessment of the entire system.

Problems found

Encapsulation violation

As it was mentioned in the previous section the tools have found 47 bugs in the code that belongs to protocol package. Those bugs were related to encapsulation violation problem. 22 classes exposed internal representation of the objects held in them by improper implemented get methods.

The flaw related to encapsulation violation problem was described in the book [11]: *"A good example of a system modification attack involves a security hole discovered in JDK 1.1.1 by the Princeton team in February 1997. The hole has since been fixed. The particular hole was a problem in the way code signing was implemented. As a result of a simple error in the JDK 1.1.1 code signing system (returning a mutable array on a Class.getsigners() method call instead of a copy of the array), an attack applet signed with a fake signature could escape the sandbox completely and acquire all the privileges available to completely trusted code. Once outside the sandbox, the attack applet could do anything at all, including installing a virus or a Trojan Horse. Put in simple terms, any machine attacked through the code signing hole could be* completely compromised."

It was found what those exposed values have influence on and how it is possible to break system's well-functioning by changing those values: can the data given or submitted by a user be tampered. During this procedure it was found that the majority of the data exposed by the classes described earlier is not used anywhere and, as a consequence, does not influence system's behavior. Also, the number of problems, such as warnings about exposing a date value, were considered as less crucial and were not further evaluated. However, a number of "real" problems were found, analyzed and further described.

Saving the voter ballot

„*PnyxApplet is a e-voting applet that acts as a local server for the web UI, which handles vote encryption, vote casting and voting receipt validation*"(comment from java class)
This applet runs a number of threads that are responsible for sending ballots with voter's voting results to the server. For the security reasons all the ballots that will be sent to the server are needed to be encrypted and digitally signed. The way encryption of the voting results happens is the main subject of interest in this section.
The class responsible for encryption and signing the vote is called ClientCrypto. To do the vote encryption this class generates EncryptionExponents object that holds the array of SecureRandom BigIntegers (it has also some other fields but they are out of our interests). These random numbers together with public key participate in vote encryption. The number of those exponents is equal to number of records in the object that holds the information about the ballot (one exponent for each record). The vote encryption itself is a calculation of two different values - one is encryption of exponent value and the second - encryption of the vote record using exponent and public key. The object that holds these values is called ElGamalEncryptionPair and represents the encrypted vote option (record) and array of these ElGamalEncryptionPairs - encrypted vote. This array together with signed original exponent values (needed to decrypt encrypted ballot using reverse formula), voter credentials (voter id and certificate) and information about election event are added to the message that will be sent to the server.

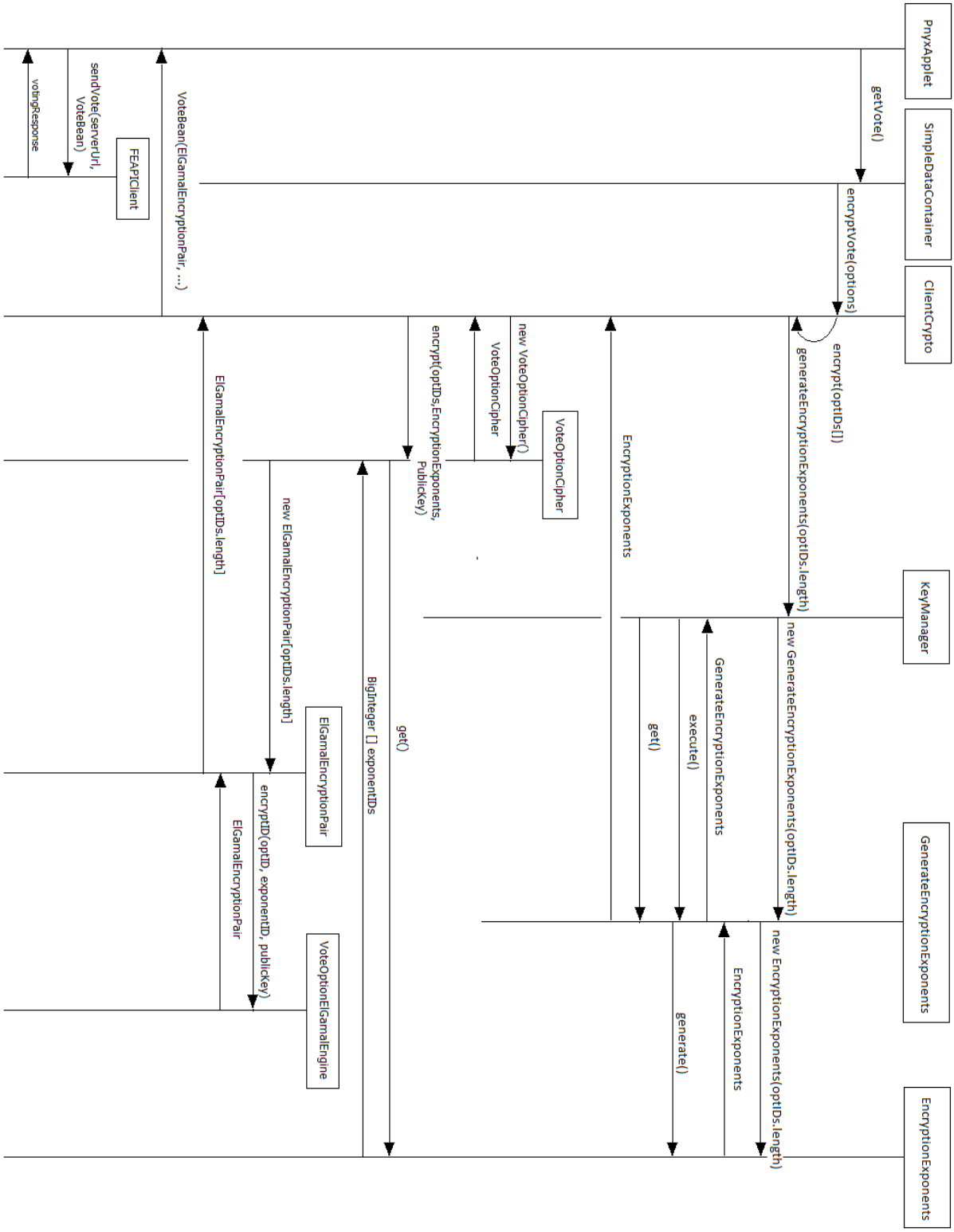

Figure 5. Sequence diagram. Encryption and saving of a vote.

Problem description

In order to modify values, set methods are usually used. However, sometimes by mistake programmers provide client's classes with the possibility to modify object's fields after retrieving them from get method. The main weakness in the process of sending a vote is that EncryptionExponents class' get method that returns encryption exponents' IDs of an EncryptionExponents object actually returns a reference to the original array. The client class gets the original array that holds those IDs, and not a new copy of it. By calling this get method, user can modify the array with exponents' IDs which is actually declared as private. EncryptionExponents class was created with the purpose to be immutable, because the set method for exponent ID values is absent. Exponents' values should be initialized only in EncryptionExponents constructor or created by generate method. There should be no way to change the values held in array of exponents' values of EncryptionExponents object manually when this object was already in use. Returning a reference to a mutable object is a potential security risk. If instances are accessed by untrusted code and unchecked changes are made to the mutable object, it would compromise security.

Solution

To eliminate this risk of exposure to untrusted client, the get method should not return the field value itself, but the copy of it. In this case the client class gets its own instance of the value and can change only it, the original value remains persistent.
In the code listed below EncryptionExponents is the same class as was described in the previous section - it has the same fields, getters and generate() method to generate array with encryption exponents' ids. EncryptionExponentsCloned has all the same properties as EncryptionExponents with the only difference - it returns a copy of array with encryption exponents' ids. In other words EncryptionExponents' get method was changed from this

```
public BigInteger[] get() {
      return _exponents;
}
```

to this

```
public BigInteger[] get() {
      return _exponents.clone ();
}
```

After initializing these two object the attempt to change its value was made

```
int numVoteOpts = 5;
BigInteger q = new BigInteger ("123456789");
EncryptionExponents _exponents = new
     EncryptionExponents(numVoteOpts, q);
EncryptionExponentsCloned _exponentsCloned = new
     EncryptionExponentsCloned(numVoteOpts, q);
```

```
_exponents.generate();
_exponentsCloned.generate();

System.out.println("Not cloned\tCloned");

for(int i = 0; i < _exponents.size(); i++){
      System.out.print(_exponents.get(i)+"\t");
      System.out.println(_exponentsCloned.get(i));
}

System.out.println("\nAfter attempt to change\nNot
                                        cloned\tCloned");

for(int j = 0; j < _exponents.get().length; j++){
      _exponents.get()[j] = null;
      System.out.print(_exponents.get()[j]+"\t");
      _exponentsCloned.get()[j] = null;
      System.out.println(_exponentsCloned.get()[j]);
}
```

The result shows that values held by EncryptionExponentsCloned object was not possible to change.

```
Not cloned    Cloned
96915198    85546340
506749        67142430
79110255    112351712
30622884    16292462
92740677    2287159

After attempt to change
Not cloned    Cloned
null    85546340
null    67142430
null    112351712
null    16292462
null    2287159
```

Return Code generation

„The voting card is a paper sheet containing a unique Return Code for each party list and for each position in the party list. Voting cards are used for the verification of the correct recording

of the voters intent (cast as intended) by the Vote Collector Server (VCS). To this end, after the voter has cast a vote, the Return Code Generator (RCG) returns the Return Codes of the parties selected by the voter, and the Return Codes corresponding to the positions of the selected candidates in the party list. Therefore, voters can verify if the Return Codes returned by the RCG match with the ones available on the Voting Card for the same selected voting options.“ [5]
Before the elections the eVoting system gets zip file that contains files with election configuration and candidate list and a copy of electoral roll from Administrative system [6]. These files are used by eVoting system to generate voting cards and return codes. GeneratePartialVotingCardCommand class is responsible for generating files with the voting cards. This class functionality is implemented as a shell command that can be executed by eVote administrator. By using files downloaded from Administrative system this class creates a contestProperties object that holds number of candidates per party, number of parties, number of vote options (referendum answers) and voter IDs, which are divided by comma.

The example:

```
parties = 2
candidatesPerParty = 2
voterIds=03015200242,03011500292,03011600122,03011700143
referendumAnswers = 2
```

The GeneratePartialVotingCardCommand class makes a number of threads and divides all voter IDs between them. Each thread runs the voting card generator for each voter id. This generator makes zipped files and stores them in the system. There are two files generated: *voterId*-BallotIdMap.ser file that holds voter id and his ballot id, and *voterId*-PartialRetCodeCollMap.ser file where ballot id with voter’s return codes for all parties and all candidates of those parties are stored.

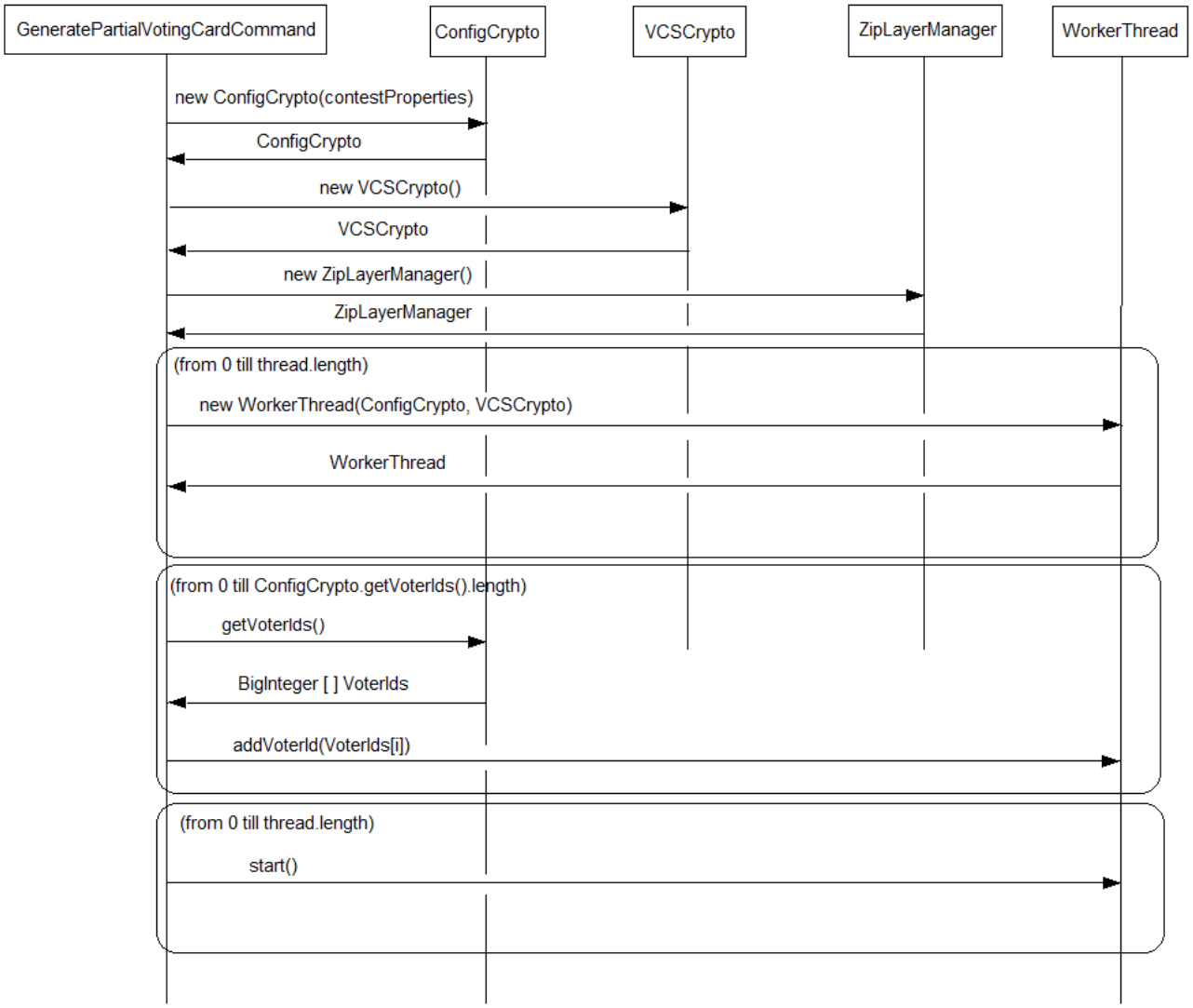

Figure 6. Making threads for voting card generation.

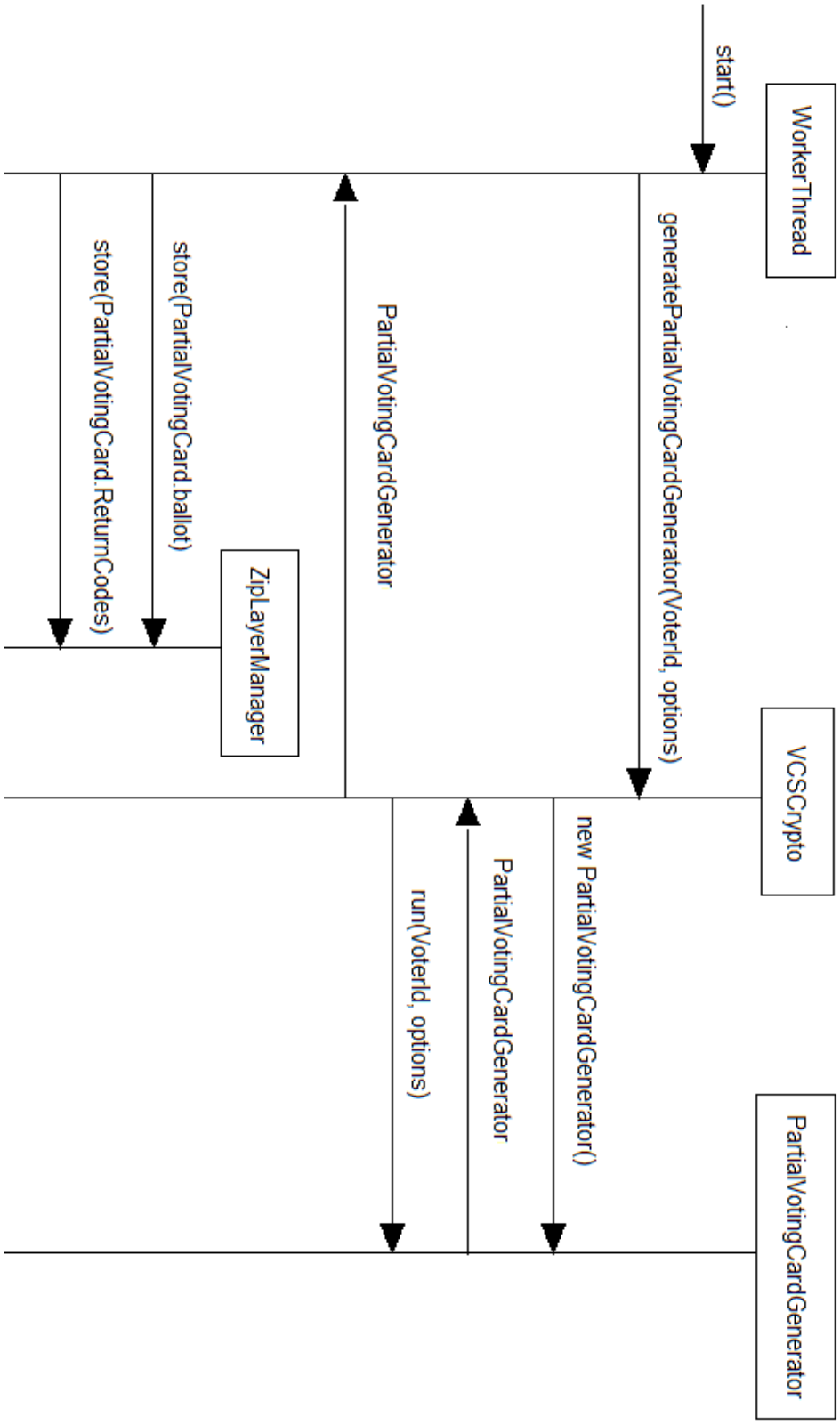

Figure 7. Voting card generation

Problem description

ConfigCrypto class gives a possibility to manipulate voter ids using its getVoterIds method. This method returns the array of all voter IDs that have been read from Electoral Roll. In case values are changed, the wrong values of ballot id and return codes will be generated or won't be generated at all.
However, if we compare the version 1.0.7 used in elections and the initial version of the system then it can be seen that there was made an improvement of the return code generation solution. The initial version the GeneratePartialVotingCardCommand class used an array with party/candidate identifier to generate return codes. The content of the array could be manipulated, in order to acquire some level of control of what return code will be generated. For example, the values in this array could be shifted, replaced or the whole content can be deleted. The problem was with using VoteOptions' get method that returned all vote options identifiers as array, providing with a possibility to change those values. In the version 1.0.7 the voting options are helt as list of ReturnCodePrimeNumbers objects instead of easily mutable array of BigIntegers.

Solution

As this problem is the same as problem described in the previous section the solution is similar: returning the copy of original array that holds the voter IDs.

Connected problems

GenerateVotingCardCommand is also implemented as shell command and is responsible for return codes generation. To do that it loads files generated by the command that we discussed above. Similar to that command class this class also works with voter ids held by ConfigCrypto class which is used by threads. Those threads load the files generated by GeneratePartialVotingCardCommand. As those filenames start from voter id, by changing voter ids value, another file can be loaded (files that belong to other voter with his ballot id and return codes, or file made and uploaded by an intruder).

Using arrays as immutable objects

Sometimes there is a need to define an object in an application as a wide constant, which is initialized only once - during the start-up. The number of those were made in the reporting package. They are

```
public static final String[] AREA_COLUMNS =
     {"electionEvent", "country", "county", "municipality", "borough",
               "pollingDistrict", "pollingPlace", "pollingStation" };
```

```
public static final String[] ELECTION_COLUMNS =
      {"electionEvent", "electionGroup", "election", "contest"};
```

These arrays hold the names of table colums and are used to built Hibernate queries to database.

Problem description

The fact that the name of the class is Constants.java and its values are initialized as finals leads to the assumption that those two objects were made to be constant throughout the lifetime of the application. The code listed below proves that these value actually can be changed.

```
System.out.println(Constants.AREA_COLUMNS[0]);
Constants.AREA_COLUMNS[0] = "****";
System.out.println(Constants.AREA_COLUMNS[0]);
```

Line 1 prints “electionEvent”. On Line 2, represents an attempt to make a change and Line 3 prints “****” whereas it was supposed to print “electionEvent”.

These values are used by ReportRepositoryImpl and TemplateRepositoryImpl classes to built quesries to tables with reports’ information. Eventhough it was not found how the class that builds the queries, it is important to fix the problems found here as attacker can use methods of these clases to try to perform other queries to database. For instance, if to change these arrays to contain only “1”s and make the client class that will call getTemplateListByElection method passing to it values that contain only “1”s then two “1=1” criterias will replace original ones.

Solution

The one possible solution would be redefining declaration of an object AREA_COLUMNS as:

```
public static final List<String> AREA_COLUMNS = Collections.unmodifiableList(
        Arrays.asList("electionEvent", "country", "county", "municipality",
                "borough", "pollingDistrict", "pollingPlace", "pollingStation" ));
```

The method that is trying to consume Constants.AREA_COLUMNS cannot do any updates now. Line 2 in the listing below will cause an “UnsupportedOperationException” – a runtime error that indicates that a particular operation – set() in this case cannot be run.

```
Exception in thread "main" java.lang.UnsupportedOperationException
  at java.util.Collections$UnmodifiableList.set(Collections.java:1156)
  at Test.main(Test.java:14)
```

Conclusion

The introduction of internet voting system has both positive and negative cases. The voters who used the system to cast their vote on the elections 2011 were satisfied with their experience and most of the norwegian citizens see the value and benefits of it for themselves. As it was expected, the e-voting increased accessibility and availability. Nevertheless it did not increase voters' turnout, but made the voting process more comfortable for persons who would have taken part in the elections anyway [14]

In the security perspective the trial can be considered as a success: no cases of well-turned attacks that could harm the election results were (officially) reported. However, there were cases of hacking attempts. While the audit of the log files was performed by Scytl, nine invalid votes passed for counting were identified. The Scytl has analyzed the possible scenarios of generation of those invalid votes and concluded that "the votes were generated by including more than one selection for the same candidate due to an attack or an applet error when casting the vote". However, "these invalid votes had been detected during the counting process and, therefore, not included in the count, (but) the voter would have believed that a valid vote had been submitted at the time of voting, as these invalid votes were accepted by the return code generator and would have led to a return code being sent to the voter." It is one case when the vote has not been counted, but the voter has received a valid return code. The other case was the situation when voter casted his vote just before his online session expiration, so the time when server received the vote was already after the session expired and vote was cleansed [15]. The interview among persons participated in e-voting was conducted, and it was found that there were cases of receiving invalid return codes. However, some people participated in the eVoting confessed that they did not checked the validity of return code, the fact that they received the SMS convinced the voters that their ballot was cast successfully [14] [15].

The implementation of the eVoting system is not perfect, and none can be. To be aware of the systems unusual behavior a strong audit system is needed. Not all the voters were willing to take the opportunity to check the correctness of their votes' casting. At some point the Ministry found out that the stackers do not intend to make use of the verification possibility and conduct an audit of the system. So, the Ministry had to hire organization to carry out the auditing. The Computas AS was chosen for this purpose [15].

Attackers will always find a way to take advantage of any weakness in the system: the flaws described in the documents that are shared on the Ministry site, the holes found during this work, lack of strong auditability or any other that has not been detected yet. In the previous sections it has been mentioned several times that the usage of some parts of the code has not been found. However, it does not mean there isn't anything to find.

The developers have done a number of correct decisions about system's functionality: practicing zero-trust to voter's machines (a voter can be compromised), the possibility to overwrite vote unlimited number of times (can help to minimize, for example, family voting), the possibility by casting p-vote to overwrite all e-votes cast before or after p-vote casting (could help to minimize selling/buying the votes, as a buyer can never be sure that e-vote he have bought will be actually counted) [16]. Nevertheless, there is always a room for improvements.

From developers' point of view the only possible attack is denial of service attack and that the the level of likelihood of hacking is low. The first argument is that the cost performing an attack is high, so there should be a strong motivation for that. The second argument is that an attacker will not be able to change that ballot or return codes in a useful way, as there is no mapping in the code between voter and voter Ids, and codes and candidates [16]. However, changing this data in some way can cause problems. Even though data tampering would be detected on the later stages, it could be a reason for cancellation and postponement of elections. The developers' main focus is on outside attacks. Their choice is justified: the external attacks are wider scale and draw more attention. Also most employees are not likely to harm their employer, however, this is not guaranteed. Internal attacks may never be discovered. The best practice is limiting employee access to areas needed to perform their job. For the problem case that was described in this work, it is worth and recommended, for example, to restrict access to the number of database tables stored in RCG. These tables are used to perform mapping between return codes and actual parties and candidates. Anyone having the access to these tables' data would know what values should be changed to tamper the results in a useful way.

References

1. 24 deadly sins of software security. Programming Flaws and How to Fix Them. Michael Howard, David LeBlanc and John Viega. The McGraw-Hill Companies. 2010
2. Countries with e-voting projects. The Electoral Knowledge Network. http://aceproject.org/ace-en/focus/e-voting/countries
3. Norwegian Ministry of Local Government and Regional Development website. www.regjeringen.no
4. Scytl official website. Customers. http://www.scytl.com
5. E-vote 2011. System Architecture. eVote. V 1.3. ErgoGroup
6. E-vote 2011. System Architecture. Overview, Interfaces and Deployment. V 1.5. ErgoGroup
7. Use case specification: 2.1 E-voting. Project: E-vote 2011. Norwegian Ministry of Local Government and Regional Development
8. What Is Java? Paul Leahy, About.com Guide. http://java.about.com/od/gettingstarted/a/whatisjava.htm
9. Java Security. Joseph A. Bank. 1995. http://groups.csail.mit.edu/mac/users/jbank/javapaper/javapaper.html
10. JavaCard Platform Security. Technical White Paper. Sun Microsystems, Inc.
11. Securing Java. Getting Down to Business with Mobile Code. Gary McGraw and Ed Felten. John Wiley & Sons, Inc., ISBN: 047131952X
12. Concepts in Programming Languages. John C. Mitchell. Cambridge University Press 2003
13. Alert (TA12-240A). Oracle Java 7 Security Manager Bypass Vulnerability. US-Cert website
14. Summary of the ISF report. http://www.regjeringen.no/en/dep/krd/prosjekter/e-vote-2011-project/evaluations-of-the-e-voting-trials/evaluation-of-the-e-voting-trials-in-201/summary-of-the-isf-report.html?id=685824
15. Speed and Efficiency of the Vote Counting Process. Norwegian E-Vote Project. Jordi Barrat i Esteve, Ben Goldsmith and John Turner. June 2012
16. Security: Zero Trust-briefing. http://www.regjeringen.no/en/dep/krd/lyd_bilde/nett-tv/E-vote-2011-Security-Zero-Trust-briefing.html?id=559446E-vote 2011
17. PMD webpage http://pmd.sourceforge.net/
18. FindBugs webpage http://findbugs.sourceforge.net/