DemTech Research

Security Analysis of Scantegrity System

DemTech Technical Report Series TR-2012-1

May 2012

Copies may be obtained by contacting:

DemTech Project
IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone:	+45 72 18 50 00
Telefax:	+45 72 18 50 01
Web	`www.demtech.dk`

This work is supported in part by DemTech grant 10-092309 awarded by the Danish Council for Strategic Research, Programme Commission on Strategic Growth Technologies.

Security Analysis of Scantegrity System

Amir Rached

Submitted in partial fulfillment for the degree of Bachelor of Science.

Committee:

- Professor Carsten Schürmann
- Professor Joseph Kiniry

Contents

1 Introduction

This thesis presents an attempt to study the gap between the implementation of Scantegrity voting system and the one described in the papers [4] [5]. This gap that could be shrunk by implementing the back-end of Scantegrity that is described in these papers. This thesis also presents a security analysis of Scantegrity that focuses on the logic, mathematics, design, implementation and deployment of its voting protocol. The main result of that analysis is that the actual implementation of Scantegrity contains many flaws at different levels.

2 Scantegrity

Scantegrity is a security component that can be integrated to the common electronic voting system that uses an optical scanner to scan voted ballots and tally them, Scantegrity is used in such a way to improve the security of the underlying voting system using cryptographic techniques. These techniques allow the optical voting system to produce receipts so that the voters can make sure that their ballots are counted and unmodified. In addition to that, anybody can check that the number of votes corresponds to the number of voters who voted. This option does not change anything in the optical scan voting procedure, it is only for the voters who want to check their votes themselves without violating the ballot's secrecy.

2.1 Basic Idea

In any voting system, with or without an electronic component, the core security problem is chain of custody.

Any attacker who breaks the procedure starting from the moment the voter cast the vote till the moment of announcing the result of the tally known as the chain of custody. This includes attacks such as adding voted ballots to the ballot box, deleting votes, switching votes or adding votes to contests that the voter left empty. This attack can be achieved by inserting malicious code or changing paper ballots, and it is undetected with manual vote recount.

The latest electronic voting systems have minimized the system's reliance on chain of custody, they provide cryptographic checks to assure that the ballots have been correctly recorded and provide the voter with receipt to check that the voter's vote has been counted correctly without revealing which candidates the voter voted for. This prevents the possibility of selling votes.

This set of voting systems is called end to end or E2E, to refer to the idea of a voter that can check that his/her vote is casted correctly and included correctly in the final count.

In these systems, any break in the chain of custody will lead to inconsistent public record which is detectable. These systems use special type of ballot in order to ensure the security of the procedure, the challenge here is to achieve that with minimal impact on the procedure the voters go through in order to vote.

Scantegrity combines the cryptographic techniques of E2E systems with the widely used vote-counting system, it does not interfere with existing procedure such as paper audit trail and manual recounts so Scantegrity has a minimal impact on election procedures and is the first independent E2E security system that keeps optical scan as the underlying voting system and does not interfere with manual recounts.In additional to that, Scantegrity provides elections with universal verifiability that allows anybody to make sure that the votes are counted correctly using mix-nets [6].

mix-nets [8] provides anonymous but verifiable connection between the voter and vote. The mix-net performs a cryptographic operation at each node to hide the path between the voter and his/her casted vote.

2.2 Scantegrity System

The ballot

The Scantegrity ballot is a normal optical scan ballot, it contains confirmation codes for every candidate, these codes are printed in an invisible ink, marking a ballot is done by revealing the confirmation codes related to the selected candidate using a decoder pen.
Every ballot contains at the end a perforated part, called receipt, where the voter can note the confirmation codes revealed for later check.
Every ballot has a unique serial number.

The optical scanner

The optical scanner uses a standard mark detection to count a vote for a candidate when the corresponding bubble is darkened by the decoder pen.

The decoder pen

The decoder pen is a special pen used to reveal the hidden confirmation codes related to the selected candidate by darkening the bubbles of those confirmation codes.

The receipt

The receipt is a perforated part in the bottom of the ballot for the voter to optionally write the confirmation codes of the selected candidate that appeared in the bubble. The receipt also contains the serial number of the ballot.

The confirmation code

The confirmation code as shown in figure 1, is a sequence of pseudo-randomly generated alphanumeric characters that needs to be unique for each contest on each ballot. The confirmation code for a particular candidate is unknown to the voter until the voter marks a vote for that particular candidate.

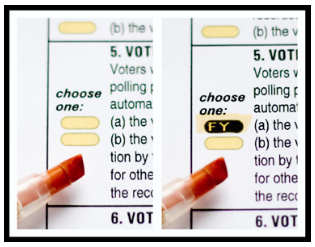


Figure 1: Confirmation code

2.3 Scantegrity Back-end

The majority of E2E voting systems can be generally divided into two parts:

- *Front-end*, which refers to the interaction between the voter and the voting system's tools and equipments in order to vote and get the receipt.
- *Back-end*, which refers to the mechanism adopted by the voting system in order to tally the ballots, publish the result and show its consistency.

The back-end describes how Scantegrity deals with the data presented such as candidates, ballots, confirmation codes and votes. How this data is organized and what is published publicly and what is secret.

It links the confirmation codes to the candidates through various tables.

Table P

Table P as shown in figure 2, Contains the correspondence between the confirmation codes and the candidate on each ballot in the order in which they were generated by the PSNG.
Row i represents ballot whose ID is i; column j represents candidate j, and the confirmation codes (i,j) belongs to the candidate j on the ballot i . Table P is never published it is just used to produce **Table Q**.
This table is generated before opening the poll and remains secret in hands of election officials because it explicitly reveals the relations between each candidate and the confirmation codes that refer to him/her on each ballot.

Table Q

Table Q as shown in figure 3, Contains each row in **Table P** but the confirmation codes are randomly permuted; that means that row i corresponds to ballot i, but a column does not correspond to a fixed candidate. This table is published on the election's website.

Table R

Table R as shown in figure 4 contains 3 columns; flag, Q-Pointer and S-Pointer. Each row of

Ballot ID	Nicolas	Rebecca	Rachelle
0001	E9F	TRK	LM7
0002	S2N	3RD	1MK
0003	6GD	HO6	KJ4
0004	U7I	R3W	QWA

Table P

Figure 2: Table P

Ballot ID			
0001	TRK	LM7	E9F
0002	1MK	S2N	3RD
0003	6GD	KJ4	HO6
0004	QWA	R3W	U7I

Table Q

Figure 3: Table Q

Table R corresponds to an underlying confirmation code from **Table Q**, and each row has a flag that will be raised in the post-election posting phase if a vote is made for that confirmation code, and also in each row exists two pseudorandom pointers: one that corresponds to the place of the confirmation code in the **Table Q**, and one corresponds to the place of the confirmation code in the **Table S**. These pointers are generated using a Pseudo-Random Generator. **Table R** is published on the election website after closing the poll.

Flag	Q-Pointer	S-Pointer
	(0004,3)	(4,1)
	(0003,2)	(1,2)
	(0002,1)	(3,3)
	(0001,2)	(1,1)
	(0002,3)	(3,1)
	(0004,1)	(2,2)
	(0001,1)	(3,2)
	(0004,2)	(1,3)
	(0002,2)	(4,2)
	(0003,1)	(2,1)
	(0001,3)	(2,3)
	(0003,3)	(4,3)

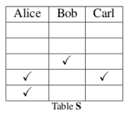
Table R

Figure 4: Table R

Table S

In **Table S** as shown in figure 5, each column j contains the confirmation codes for a candidate j, Each element corresponds to an underlying confirmation code. **Table S** is published on the election website after closing the poll.

Once **Table S** is published anyone can get the number of votes for a particular candidate by counting the number of votes marked in his column in **Table S** and then compare that to the number announced by the election officials.



Alice	Bob	Carl
	✓	
✓		✓
✓		

Table **S**

Figure 5: Table S

After closing the Poll, the election officials post a list of all the voters who voted and the tally from the optical scanner, then they use the tally and the **Table P** to translate the votes into the correspondent confirmation codes.

They use the commitments in **Table Q** to flag the entries in **Table S** and **Table R**. Anyone can count the number of votes for a particular candidate by counting the number of raised flags in his/her column in the **Table S**, these tallies are also checked against those of the optical scanners. For each row in **Table R**, either the Q pointer or the S pointer is revealed depending on a true random based on a public coin flip. The election officials reveal all the information concerning the audit ballots or the spoiled ones.

2.4 Voting Experience

The procedure is the same as in optical scan, after the authentication of the voter at the polling place. The voter takes the ballot and each voting response location contains a random confirmation code written in invisible ink, each voter who wish to vote marks the bubble related to the chosen candidate and that causes the bubble to darken and the confirmation codes to appear in that bubble. Then the voter puts the ballot in the optical scanner to read the ballot ID and the state of the bubbles.

There are two ways for a voter to verify the correctness and the honesty of the voting procedure, each voter can choose to use either one of them, both or neither.

The first way is that each voter has the right to claim an additional ballot called ”audit ballot” this ballot is stamped at the polling place ”audit ballot” so the voter takes the ballot, reveals all the confirmation codes on the ballot using the special pen then puts it in the optical scanner and takes the ballot outside the polling place. After closing of the voting, election officials publish the confirmation codes that were revealed on the scanned ballots together with the ballots IDs so the voter can check that all the revealed confirmation codes for the audit ballot exist on the ballot.If not the voter can file a dispute against the election officials claiming the incorrectness of the voting procedures using the audit ballot as evidence of that.

The second way to verify the correctness of the voting procedures is using the voting ballot itself. Every ballot contains at the end a perforated part called ”ballot chit” or ”receipt” where the voter can note the confirmation codes revealed on the ballot before putting the ballot in the optical scanner. This part of the ballot contains a copy of the ballot ID. After closing the poll, election officials publish the confirmation codes that were revealed on the scanned ballots together with the ballots IDs so the voter can check that all the revealed confirmation codes for the voting ballot exist on the ballot.If not, the voter can file a dispute against the election officials claiming the incorrectness of the voting procedures using the receipt as evidence of that.

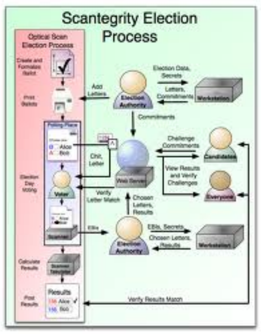


Figure 6: Overall Scantegrity Process

2.5 Security

Scantegrity provides a high-level sketch of several common attacks and the countermeasures it uses against them. In fact, the model presented assumes that the adversary could be anyone a party, a voter, a poll worker or an election official whose goal is to attack the integrity of the election or perform an influence over the voter.

The attackers could attack the integrity of the election, the privacy or the ballot secrecy.

2.5.1 Integrity

Refers to the ability of the system to detect any attempt from an adversary to modify votes by modifying ballots, creating fraudulent ballots or changing the tally on the election website.

Adding and Deleting votes

Adding votes is an extremely difficult process because an adversary will have to add names to the list of voters who already voted, also will have to change the number reported by the polling sites; an attempt like this should be easily detected by anyone.

Deleting votes by making them spoiled for auditing and posting all the information related to that ballot is also difficult because there will be a risk that the voters of those ballot file disputes that their ballots have been posted incorrectly, and they will use their stamped receipts as proof.

Adding more marks to a ballot is useless because the optical scanner will refuse any over-marked ballot, but if the attacker marks an under-marked ballot, that will be a problem so the solution to that is to add a ”none of the above” choice on each race for a voter to submit an undervote.

Fraudulent Ballots

There are several types of fraudulent ballots, one type is that there exist on the ballot several candidate with the same confirmation code, another type is that the confirmation codes printed on the ballot do not corresponds to the P table or Q-pointer or the S-pointer do not map it correctly to the selected candidate, another type is that the ballot is completely invalid such that the confirmation codes on the ballot is not similar to those committed to.

All these types can be detected using the cut-and-choose auditing of the ballots if a random half of the ballots are audited, no more than a few fraudulent ballot will go undetected.

Modified Ballots

An adversary tries to modify the flags the Q , R or S tables so the counting of the flags in the S table be incorrect can be easily detected using randomized partial checking of the Q and S pointers in the R table. The probability that modification in k pointers go undetected is $(1/2)^k$.

2.5.2 Privacy

Refers to the ability of the system to prevent an attacker to link a vote to a particular candidate to the voter who votes it.

Randomized partial checking

For each code either the Q-pointer or the S-pointer is revealed thus the randomized partial checking of the pointers hides each voter's vote equally among a randomly selected half of the votes.

Receipts

Because confirmation codes are randomly assigned to candidates and independent across ballots and the commitments function is computationally hiding, then the receipt and the confirmation code written on it cannot reveal anything related to the candidate the voter voted to.

To violate the voter's privacy, the adversary must have the knowledge of the Ballot ID that the voter will get; the codes on that ballots and the private information of the tables then the adversary will be able to affect the voter's vote.

To reinforce the privacy precautions, the ballot ID that exists on the cast part could be encrypted or not human-readable.

3 Implementation

The working version of Scantegrity is found on the repository http://scantegrity.org/svn/. The project needs to be compiled using the maven build automation tool. It has no documentation on how it compiles or runs. Professor Filip Zagorski and Professor Richard Carback provided hints how to make the system work.

The project is written in Java Programming Language, it is a set of Java classes distributed in different folders. The source code contains around 40000 lines and it is almost written in an imperative style containing 502 static methods.

The project is run through an interface where the user has to determine the path to the input files.

4 Deployment

Implementing a voting system is not an easy task for many reasons. Most of the voters are not trained and also elections do not happen so often that's why most of the voters prefer not to change the way they are voting and that issue must be taking into consideration. Also the privacy requirements prevent the usage of the common feedback and auditing routines used in other applications like banking. Also governmental laws and already existing traditions in the conduct of elections are difficult to alter.

The working implementation version of Scantegrity is the one used for Municipal Election at Takoma Park in 2009 [2] and it was considered as the first E2E binding governmental election with ballot privacy.

We have to mention that Scantegrity is the result of an intense study of other voting systems that has been successively developed, tested and deployed by a team of researcher. The closest voting systems to Scantegrity is the Punchscan voting system [3] that was developed before Scantegrity, but the problem with Punchscan is that it was never used in public elections, in additional to that it does not support multiple ballot style also it was never targeting a wide cross-segment of people as in Takoma Park, moreover the Punchscan team was more concerned about administering the elections, in additional to that Punchscan did not allow manual recount although it uses paper ballots, but manual recount was an important requirement in Takoma Park elections.

Despite all the disadvantages of Punchscan, it was the closest voting system to Scantegrity in fact this implementation of Scantegrity is built on top of the Punchscan back-end. The reason for that is, implementing the Scantegrity back-end would take a lot of time and effort and also the Punchscan back-end is more efficient since less commitments are needed.

4.1 Election Setting

The election was implemented as a series of events, each event is called ”a meeting” where election officials meet and and perform a series of actions.

4.1.1 Meeting 1

The goal of this meeting is to set a group of trustees for the coming events of the election, and to decide a threshold of trustees needed to conduct the remaining parts of the election. this threshold of trustees is enough to reveal the correspondence between the confirmation numbers and codes. So it is of extremely importance to keep the passphrases that the trustees supply during this meeting secret.

In this meeting, The trustees generate the commitments to the decryption path of all the ballots that will be used in the election and publish these commitments.

4.1.2 Meeting 2

The goal of this meeting is to create the electronic ballots and audit them. Trustees check the data at random locations and use Scantegrity code to confirm that these checks do not reveal an error. at this point, trustees check half of the ballots decryption paths committed in Meeting 1. Also they generate ballots that are correspondence between confirmation codes and also generate commitments to them. The ballots ids that are checked are published together with the results of the check.

4.1.3 Meeting 3

The goal of this meeting is to publish the results.The meeting will allow a tally to be calculated and posted, as well as posting the status of each ballot in the election (voted, provisionally voted, spoiled, or unused/audited for printing errors). For voted ballots, the confirmation codes will be posted online so voters can verify they match what they selected in the voting booth. Since codes are randomly assigned, knowing someone else's code doesn't allow you to determine how they voted.

Meeting 3b
If necessary, it will add to the tally the votes from any provisional ballots deemed acceptable.

4.1.4 Meeting 4

The goal of this meeting is to perform the final audit that is used to resolve any complains of the voters that the confirmation codes posted are not similar to what they wrote down.After doing this, the commitments that were generated in meeting one are again checked against the published confirmation codes to make sure they are consistent and this check require a random selection as in meeting two.

4.2 Back-end Architecture

In order to be able to combine the Punchscan back-end with Scantegrity front-end, the coded-vote approach was used by the Scantegrity team in this implementation. In Punchscan a "coded vote" is represented by the hole marked by the vote. If the top most hole is marked, this corresponds to a coded vote of 0 (zero), if the second top hole is marked, this represents a coded vote of 1 (one), third hole 2 (two), etc.

In Figure 7, the first choice is the only one marked so this question will have a coded vote of 0, if the first two choices were marked, the question would have coded vote of 0 1.

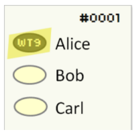


Figure 7: Coded vote

The Punchscan back-end uses 3 tables in order to perform the needed tasks to publish the results. The tables are; P table, D table, R table.

Each table is defined by attributes, we use the same attributes names as in the source code.

The random salts used to generate the commitments are stored in the tables in order to minimize the amount of data stored after each meeting.

P table

A single P table is used, for every election. Every row in the P table correspondents to a ballot.

Each row in the P table has the following attributes. These attributes are not necessary to be present all. some of them can be missing according to the situation.

Each row in the P table represents a specific ballot, the column p1 contains the information about the questions on the ballot and the number of answers available to each question, the column p3 contains the information for the maximum number of answers that can be chosen for each question.

id : is an integer from 0 to number of ballots -1. It uniquely identifies each row in the P table.

p1 : is a concatenation of permutations. e.g. consider a tuple in the table where p1 is of the form "1 0 2 1 0 1 0 3 2" and this election has 3 questions; first questions has 2 total answers, second question has 3 total answers and third question has 4 total answers that means the the first two elements of p1 represent a permutation of length 2 and the next three elements of p1 represent a permutation of length 3 and the next 4 elements of p1 represent a permutation of length 4. Moreover that an example of a permutation like "2 3 1 0" is a short form of $(0 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 1, 3 \rightarrow 0)$
this means that the first index is a consecutive index starting with 0 and the second number is where this index gets permuted to.
The maximum number of candidates that can exist is 128 so the biggest number that can exist in p1 is 127 and since there is no delimiter between questions then the length of each permutation is determined by the number of questions and the total number of answers for each question. In Punchscan, p1 represents the permutation of the top page of the ballot, but in this context it is just a concatenation of permutation.

s1 :is a pseudo-random salt used for computing the commitment to p1. It is a base64 representation of the a byte array of length 16.

c1 :is the commitment to p1 using salt s1. It is a base64 representation of a byte array of length 64. To commit to p1, the message is formed as follows: the first bytes are byte representation of the "id" represented as a string the next bytes are a byte representation of "p1" where each number in ”p1” is represented as a single byte, in the order they appear in ”p1”

p2, s2, c2 are similar to p1, s1, c1 respectively. In Punchscan they refer to the permutation in the bottom page of the ballot.

p3 : is a concatenation of coded votes. If the voter makes less than the maximum number of selection for a question (an undervote), the non-selection is represented by a -1. The order of coded votes is crucial when it comes to ranking voting. (e.g. consider a tuple in the table where p3 is of the form ”0 2 0 -1 0 1 3” and this election has 3 questions; first question has a maximum of one answer, second question has a maximum of 3 answers and third question has a maximum of 3 answers then the first element of p3 represents the coded vote of the first question, the next three elements represent the coded votes of the next question and the last three elements represent the coded votes of the last question. if the second question was of type rank that means that the first choice was coded vote number 2, the second choice was coded vote number 0 and there is no third choice selected)

page : is always “NONE” and should be ignored for Scantegrity (it represents which page was chosen by the voter if a Punchscan ballot is used).

D table

An election can be composed of one or more partitions. A partition is a set of one or more question that forms something like a miniballot that is treated as an independent election.The reason is to avoid the Italian attack (correlation between clear text votes for different races). A number of D tables is created for each partition. the number of D tables is specified in the election specifications.

each D table has the following attributes

id : Is an integer from 0 to number of ballots -1.It uniquely identifies a row in the D-table

pid: The id of the ballot in the P table. It uniquely identifies a row in the P table, it is a pointer to the P table

d2 : is a concatenation of permutations.The permutations are for the questions of the same partition. (e.g. consider a tuple in the table where d2 is of the form ”1 0 2 2 0 3 1” and this D-tables is for partition 1, partition 1 has questions 1 and 2 and the first question has a total of 3 answers and the second question has a total of 4 answers.Then the first 3 elements in d2 represents a permutation of length 3 for the first question and the next 4 elements is permutation of length 4 for the second question.)

sl :(salt left) is a pseudorandom salt used for computing the commitment to d2. It is a base64 representation of the a byte array of length 16.

cl :(commitment left) is the commitment to d2 using salt sl. It is a base64 representation of a byte array of length 64. To commit to d2, the message is formed as follows: the first byte is the partition id represented as a byte (”1” is represented as byte x=1); the second byte is the instance id represented as a byte; the next bytes are byte representation of the ”id” of the D row represented as a string the next bytes are byte representation of the ”pid” represented as a string the next bytes are a byte representation of ”d2” where each number in ”d2” is represented as a single byte, in the

order they appear in “d2”.
rid :is an integer from 0 to number of ballots -1 and it uniquely identifies a ballot in the R table (this is D5 in PunchScan). It is pointer into the R table.
d4, sr, and cr are similar to d2, sl, and sl respectively. cr uses rid instead of pid to compute the commitment.

d3 :is a concatenation of coded votes. It only gives information about questions in the current partition. e.g. consider a tuple in the table where d3 is of the form ”0 1 2 1 3 0” and the current partition is partition 1 that contains question 1 and question 2. Question 1 has a 3 maximum number of answers and question 2 has 3 maximum number of answers.then the first 3 elements of d3 represent coded votes for question 1 and the last 3 elements represent coded votes for question 2. d3 is computed as the corresponding p3 (as indicated by the pid) composed with d2. e.g.for partition 1, instance 0, id=14, pid=3 p3 is of the form ”1 0 2 1 -1 1 -1” (from the P table, the row with id 13) d2 is of the form ”2 1 0 1 0 2 3” we get d3 is of the form ”2 0 1 -1 0 -1” because:

- Partition 1 contains question 1 and question 2, that have 3 and 4 answers in total and the voter is allowed to choose maximum 3 candidates for each of them.
- p3 gets stripped down of the first elements, which correspond to the coded votes for question 0 because question 0 is not in partition 1, Thus p3 becomes “0 2 1 -1 1 -1”
- Now p3 is decomposed with d2, per question
  i. For question 1, “0 2 1” is composed with “2 1 0” and we get: zero goes to zero, zero goes to two, thus zero goes to two; one goes to two, tow goes to zero, thus one goes to zero; two goes to one, one goes to one, thus two goes to one. Thus the result is “2 0 1”, the first three elements in d3.
  ii. For question 2, “-1 1 -1” is composed with ”1 0 2 3”. Note that the sizes of the two “permutations” may (and in most cases is) not be the same. Zero goes to minus one, minus one always get unchanged, thus zero goes to minus one; one goes to one, one goes to zero, thus one goes to zero; two goes to minus one, minus one always gets unchanged, thus two goes to minus one. The result ”-1 0 -1” are the last three elements in d3.

SIDE : can be “LEFT, RIGHT, NONE, BOTH” and it indicates how the auditor wants the punchscanian mixnet audited for one particular row in the D table. If the auditor wants LEFT, the mixnet must open the commitment to the pair (pid, d2), if RIGHT, it must open the commitment to (rid, d4). NONE and BOTH do not explicitly appear anywhere (but are used internally).

R table

Each partition has only one R table, it can has one ore more D tables. So all D tables for the same partition connect to the global P table and a single R table for that particular partition.

The R tables have clear text votes so they can be tallied by anyone. The questions on the ballot are divided into partitions so the correlations between questions on the same ballot are lost but the correlations within a single question are still possible.

Each row in the R table has the following attributes that may not be all present together.
id : is a integer from 0 to number of ballots -1 that uniquely identifies a row in this R table.

r : is concatenation of text votes. The clear votes are related to the questions of the same partition.(e.g. consider a tuple in the table where r is of the form ”0 2 1 1 0 -1” id = 6 and the current partition is 1 that contains questions 1 and 2. Question 1 has a maximum number of answers of 3 and question 2 has a maximum number of answers of 3. then the first 3 elements of r represent clear text votes of question 1 which has the first choice as candidate 0, second choice as candidate 2 and third choice as candidate 1. The last 3 elements represent clear text votes of question 2 which is a multiple choice question and the first two candidates were chosen.So the voter of this ballot has made less selections on the ballot than it is expected, in that situation the ballot is called an ”undervote”.)

5 Evaluation

It is obvious now the gap between the Scantegrity back-end that is described in the papers and the one that is already implemented and was used, Scantegrity back-end that was described in the papers uses 4 tables P,Q,S and R. the content of these tables is the confirmation codes on the ballots. But the Punchscan back-end that was used in the implementation of Scantegrity uses 3 tables P,D and S. It uses commitments , salts and permutations of clear text votes and coded votes. However Scantegrity has more flaws both on the mathematical level and the implementation level, in this section, we will try to point out some of these flaws.

5.1 Douglas Wikstroem Attacks

Douglas Wikstroem and Shahram Khazaei published a paper that proposes several attacks to break the correctness and the privacy of mix-nets both homomorphic and chaumian [9]. Since the majority of Scantegrity mix-nets are implemented using chaumian mix-nets, we are interested in the attacks on that type of mix-nets. These attacks are bases on two things:

1. Mix-servers that do not make sure that there is no duplicates in its inputs.

2. Mix-servers do not check that all commitments (revealed and unrevealed) are consistent with a permutation.

5.1.1 The Duplicates Thread

Jakbsson et al. [8] did not mention that each mix-server must make sure that the input that it gets has no duplicates. the presence of duplicates in the input could make the mix-net operations vulnerable against any attack.

Consider an adversary that corrupts $s(s+1)/2$ senders and the first and the last mix-servers, and that adversary is concerned about the first s ciphertexts $c_{1,0}$,.....,$c_{s,0}$. For each ciphertext, the adversary peels off the first layer of encryption using the private key of the first mix-server then the adversary makes i independent encryptions of $c_{i,0}$ using the public key of the first mix-server, thus the adversary would make $s(s+1)/2$ corrupted ciphertexts that each one of the corrupted ciphertexts would send to the mix-net. After doing this, each ciphertext i would have $i+1$ related ciphertext by recording the output of the last mix-server the adversary can identify each ciphertext by the number of its copies, and since the last mix-server is corrupted , the adversary can reveal the plaintext of each ciphertext coming from the targeted senders. As $s + s(s+1)/2 \leq N$ must hold, this attack can break privacy of at most *O* *(*$\sqrt{N}$*)* senders.
This problem can be avoided if each mix-server discards duplicates in its input or if a CCA2 encryption scheme is employed and that can be done by ensuring that the first mix-server is not corrupted.

5.1.2 The Duplicates Thread and The Inconsistency of Opened Commitments with a Permutation Thread

An adversary can corrupt the first sender and the first mix-server and then the adversary replaces all the submitted ciphertexts of all senders with the encrypted text of the first sender $c_{1,0}$ That list of identical inputs is processed and permuted correctly by the first mix-server and the output list is produced *(*$c_{1,2}$,......,$c_{N,2}$*)*. So this would be an evidence for the first mix-server to prove that it has corrupted its input. Therefore the output list of the last mix-server is *N* encryptions of *m*. To make the attack more realistic , the adversary can submit several different plaintexts and choose a suitable distribution over these and apply the attack for each plaintext. This attack is undetectable except with manually inspection the list of decommitted integers.

5.1.3 The Inconsistency of Unrevealed Commitments with a Permutation Thread

Even if the duplicates are removed at each mix-server and all the open committements are proved to be consistent with a permutation but there will be no guarantee that the unrevealed committements contain distinct integers.
If an adversary can corrupt the first sender and the first mix-server and change a ciphertext of an honest sender with a copy of the ciphertext of the corrupted sender, this attack is only detected if both ciphertexts are revealed, so the probability that this attack is detected is $\frac{1}{4}$, if *t* ciphertext are replaced independently than the probability that this goes undetected is $(\frac{3}{4})^{(t)}$ For Chaumian mix-net the replacements corresponds to replacing the final plaintext if the last mix-server in the chain makes the replacements; otherwise, since the duplicates are removed, replacing ciphertexts results in eliminating plaintexts from the final mixed output.

If this attack is repeated *q* times then the probability that this goes undetected is $1-(1-(3/4)^t)^q$

5.1.4 Universal Verfiability When Checking Is Performed at the End of the Mixing

When the check is done in-phase with the mixing, universal verifiability is weaker because this allows the adversary to change a certain number of ciphertext at each mix-serve, so the probability of success for changing *kt* ciphertexts is $(1 - (1 - (3/4)^t)^q)^k$ but this holds if duplicates are not removed. If duplicates are removed then the ciphertexts will be eliminated except for the last mix-server.

If we choose not accept the weaker guarantee of the in-phase checking, then we have to perform checking after the mixing, but doing so reveal a great thread that an adversary can just corrupt the first mix-server and change some ciphertext and then it will be caught but by that time, the votes of the senders whose ciphertexts have been changed will be revealed. To solve this problem, Douglas and Shahram proposed that the ciphertexts must be protected with an innermost cryptosystem whose secret key is shared between all the mix-servers and only decrypted after the checking is done.This can be achieved by making each mix-server generate another pairs of keys and let the joint key be the list of all the additional public keys and this prevent us from using a costly key generation protocol but it has two problems first, increased the size of the ciphertexts and second using this scheme, the execution can only be aborted if cheating is detected.

5.2 Initial Automated Language-based Security Analysis

The implementation of Scantegrity discussed by this thesis showed the existence of many bugs during the security analysis, some of these bugs are regarding correctness other are caused by bad practice. In this section, some of these bugs will be pointed out and their locations in the code will be revealed. The tools used to find these bugs are metrics [1], pmd, checkstyle, findbugs [10] [7] plugins to eclipse, these are simple static syntactical tools.

Dereferencing the result of readLine() without a null check

This bug exists thrice in Drills.java in the same method `read(BufferedReader in)` that is a public method and returns a Vector$< Point2D.Float >$.

This method will give a null pointer exception if the `BufferedReader` it reads from is empty or containing only one line.

This method is called twice in another public method called `DrillFilesToPdf(String inDir,String background, String outFile)` taking as a parameter the content of the String `inDir` concatinating to it "/drill.txt".

If an adversary managed to call the public method `DrillFilesToPdf(String inDir,String background, String outFile)` with a path to an empty file called drill.txt or a fill with just one line with the same name, this will cause a crashing error.

Field only ever set to null

This bug exists in RecordedAsCast.java

1. The variable `outDir` is set forever to null and that will cause a `FileNotFound` Exception when the method `recreateVotedBallot(String serial)` which is a public method in the class RecordedAsCasted.

   So if an adversary managed to insert bad code that calls this method, then the Exception mentioned above will be thrown and the system will crash.

2. The variable `es` is set forever to null and that will cause a `nullPointer` Exception when the method `recreateVotedBallot(String serial)` which is a public method in the class RecordedAsCasted.

   So if an adversary tries to call this method or tries to call the main of the class, the exception mentioned above will be thrown and the system will crash.

3. The variable `es` is called again in method `addSymbols(String meetingThreeOut)` so calling this public method from anywhere in the code will cause a `nullPointer` Exception. and the system will crash, This method is in RecordedAsCasted class.

<u>Non-virtual method call passes null for nonnull parameter</u>

This bug is in IRVContestResult.java calling the empty constructor in this class calls the other constructor with parameters `(-1,null)`, this constructor calls the method `setCandidates(Vector<Contestant> p_contestants)` with a parameter null, this method is tries to initialized a new vector with a null parameter which will cause a `nullPointer` Exception.

If an adversary is tries to call the empty constructor, then the system will crash.

<u>Possible null pointer dereference</u>

1. This bug is in BallotStorePanel.java line 476.

   The method call `getErrorBallotCount()` invoked on the object `c_errorResolver` must be preceded by a null check because if the object is null this will lead to a `nullPointer` Exception which is a crashing bug.

   So if an adversary succeeds in calling the public method run from the LoaderThread class, then the Exception mentioned above will be raised.

2. This bug is in ParseM2CheckCommitments.java line 40

   The public method `startElement(String namespaceURI, String lName, String qName, Attributes attrs)` contains a call for the method `getPid()` in the class BallotRow.java but the object on which the method is invoked can be null if the last parameter in the call for the method `startElement(String namespaceURI, String lName, String qName, Attributes attrs)` contains something wrong.

so if an adversary managed to do that there will be a `nullPointer` Exception which will cause a crashing error.

3. This Bug is in GenerateDummyBallots.java line 101

   Calling the public method `generateOneBallot(ByteArrayOutputStream baos)` with a bad parameter will crash the `ObjectInputStream ois` and the catch public will be executed and then the next line will cause a `nullPointer` Exception because it will try to access the variable `localES` which will be null.

   So if an adversary inserts a bad code to call this method with a bad parameter then a null pointer exception will be thrown and the system will crash.

6 Conclusion

To conclude, this thesis tried to study the gap between the description of Scantegrity system and its implementation and found that this gap can be shrunk by implementing the back-end of Scantegrity, also it showed the existence of several bugs using a static syntactical analysis performed by simple tools. It is the first step to more studies that will investigate the effect of Douglas's attacks on the implementation of Scantegrity and will use more advanced tools to detect more critical bugs that can be in the source code.

References

[1] Thorsten Arendt and Gabriele Taentzer. Integration of smells and refactorings within the eclipse modeling framework. In *Proceedings of the Fifth Workshop on Refactoring Tools*, WRT '12, pages 8–15, New York, NY, USA, 2012. ACM.

[2] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity ii municipal election at takoma park: the first e2e binding governmental election with ballot privacy. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[3] R.T. Carback and Baltimore County. Computer Science University of Maryland. *Security Innovations in the Punchscan Voting System.* University of Maryland, Baltimore County, 2008.

[4] David Chaum, Richard Carback, Jeremy Clark Aleks, Er Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-

to-end verifiability for optical scan election systems using invisible ink confirmation codes abstract.

[5] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-end voter-verifiable optical- scan voting. *IEEE Security and Privacy*, 6:40–46, 2008.

[6] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[7] David Hovemeyer and William Pugh. Finding bugs is easy. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, OOPSLA '04, pages 132–136, New York, NY, USA, 2004. ACM.

[8] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353, Berkeley, CA, USA, 2002. USENIX Association.

[9] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. *IACR Cryptology ePrint Archive*, 2012:63, 2012.

[10] Nick Rutar, Christian B. Almazan, and Jeffrey S. Foster. A comparison of bug finding tools for java. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ISSRE '04, pages 245–256, Washington, DC, USA, 2004. IEEE Computer Society.