Software vulnerabilities in the Brazilian voting machine

Diego F. Aranha[1], Marcelo Monte Karam[2], André de Miranda[2], Felipe Scarel[2]

[1]Department of Computer Science – University of Brasília
[2]Center of Informatics –University of Brasília

Version 1.0.1

[1]Team coordinator.

Abstract

This report presents a security analysis of the Brazilian voting machine software based on the the experience of the authors while participating at the 2nd Public Security Tests of the Electronic Voting System organized by the Superior Electoral Court (SEC). During the event, vulnerabilities in the software were detected and explored to allow recovery of the ballots in the order they were cast. We present scenarios where these vulnerabilities allow electoral fraud and suggestions to restore the security of the affected mechanisms. Additionally, other flaws in the software and its development process are discussed. In particular, this report details the main design and/or implementation problems detected on the security mechanisms of the voting machine software:

- **Inadequate protection of ballot secrecy:** votes are stored out of order, but it is trivial to recover them in order only from public products of an election and superficial knowledge of the software source code, which is also made public to the political parties.
- **Inadequate use of encryption:** the same encryption key is shared among all voting machines for encrypting their memory cards. Using the classical abstraction of a locker as an encryption technique, this is equivalent to using half a million lockers with exactly the same key, since this is the approximate number of voting machines in operation. This cryptographic key is also stored in the plain text portion of the memory cards. Using the same analogy, this is compatible to hiding the locker key under the carpet and trusting the secrecy of this location to protect the confidentiality of the key.
- **Obsolete cryptographic algorithms:** the cryptographic hash function used for computing digital signatures and integrity checks is demonstrably not collision-resistant. These specific applications of the chosen hash function are not recommended for 6 years already.
- **Inappropriate attacker model:** significant emphasis is put on the design of security features resistant only to outsider attackers, when insider threats present a much higher risk.
- **Faulty software development process:** bad engineering practices allow the accidental or malicious insertion of software vulnerabilities, clearly attesting that the software development process is immature from a security point of view.

- **Insufficient integrity check**: the voting software checks its own integrity during its initialization process, but all of the information to subvert this verification is contained inside the voting machines, with different attack surfaces depending on the presence of a hardware security module. In the machines without this module, the problem of software authentication is reduced to itself, with no external source of trust. In this case, digital signature-based software self-verification [1] is equivalent to trusting the authenticity of a document based only on the testimonial of the "author", who is free to impersonate anyone. It is also important to emphasize that an authentic signature confirms only the processing of the protected object at a point in time and space where the signing private key was also present. Even when the integrity verification mechanisms are not circumvented, digital signature techniques can not attest that software is in fact correct or secure. Digitally signing vulnerable software also has the opposite effect of providing mathematical certainty that all of the voting machines have the same exploitable flaws. The version of the source code studied by the authors also had commented out a function call to perform integrity verification of a significant portion of the voting software, directly illustrating the intrinsic limitations of the technique.

Detailed descriptions of the problems mentioned above are presented in the rest of this document, but it can be noted that many of the protection features implemented in to voting machine software aim to achieve *obfuscation* instead of *security*, not resisting to insider attacks or persistent threats. Several of these problems are the result of architectural flaws or inappropriate design assumptions. Fixing the underlying causes will require more than *ad hoc* localized interventions in the source code. A complete review of the software development process is needed to establish good engineering practices and avoid the intentional or accidental insertion of new vulnerabilities by internal or external attackers. Since the Direct Recording Electronic (DRE) voting machines adopted in Brazil require software integrity to provide integrity of results, the problems discussed in this report achieve a critical status and require the introduction of software independent auditability measures. Only with permanent and scientific evaluation, it is possible for the Brazilian voting system to satisfy minimal and plausible security and transparency requirements.

Contents

Chapter 1

Introduction

The Brazilian electoral authority (SEC) has been increasingly adopting electronic elections since 1996, culminating to the current scenario where a considerable fraction of the voting machines have fingerprinting devices for voter identification. Important milestones in the history of the initiative were the first purely electronic elections in 2000, the transfer of full responsibility for the software development to the SEC in 2006 and the migration to the GNU/Linux operating system in 2008. When software components and human procedures for the voting process become stable, security testing is the next natural step to improve reliability of elections and reassure that the system provides sufficient ballot secrecy and integrity.

An important movement in this direction are the public and periodic testing of the voting systems organized since 2009. Despite some undesirable restrictions, these tests allow teams of specialists from industry and academia to independently evaluate the security mechanisms adopted by the voting system.

1.1 Goals

The main goal of this report is to present the observations collected by the authors during their participation on the 2nd edition of the Public Security Tests organized by the SEC. The official reports of the event were jointly written with the SEC and do not contain sufficient information regarding other security issues not directly attacked by the authors during the event. Our intention is to point several limitations of the Brazilian electronic voting system and to contribute to its security process. Following standard practices in the security industry, we present self-contained descriptions of the observed software and development process flaws with multiple suggestions for correction or mitigation. This way, the interested parties are in an adequate position to implement effective counter-measures.

This report discusses only aspects of the voting machine software, not

Figure 1.1: Brazilian voting machine and its two terminals. The election officer terminal is on the left and the voter terminal is on the right.

discussing physical or hardware aspects of the equipment to respect the authors' fields of expertise. The information provided only pertains a small – yet strategic – fraction of the software source code, excluded other software components that constitute the complete voting system, since the rules of the event and time restriction imposed on the investigators did not allow for a full evaluation. Content is entire responsibility of the authors and does not necessarily represent the position of University of Brasília or any other institutions where the authors have worked or will work in the future.

1.2 System overview

The Brazilian voting machine is a classical Direct Recording Electronic (DRE) device without a Voter Verified Paper Audit Trail (VVPAT). It consists of an election officer terminal used to authenticate electors by their registration number or fingerprint and a voter terminal where votes are cast. Both terminals are connected by a cable and shown in Figure 1.1. In general terms, an election using the voting machine follows the preparation steps below:

1. Production of software components and distribution of memory cards containing the voting software.
2. Installation of the software in the voting machines from the memory cards.
3. Distribution of the machines to the corresponding voting places.

On the day of elections, each voting machine executes a well-defined procedure:

1. Printing of the *zeroth*[1], official public document which supposedly attests that no votes were computed for any candidates before the start of the elections.

2. Opening of the voting session by the election officials.

3. Granting of access for electors to cast their votes in the voting machines.

4. Closing of the voting session by the election officials.

5. Printing of the Partial Summation (PS) by each voting machine.

6. Recording of authenticated public products of the elections, consisting of a digital version of the PS, a chronological record of events (LOG) and the Digital Record of Votes (DRV).

7. Authorized violation of the seal by the election officials and retrieval of the Memory of Results (MR), an orange USB drive containing the public products of the election.

8. Transmission of the public products to the totalization system through a private network.

The role of the totalizator is combining all the partial summations to declare the global result of the elections.

1.3 Organization of the document

The report obeys the following structure. Chapter 2 briefly describes the format and the results obtained in the Public Security Tests. Chapter 3 details the progression of vulnerabilities which provided a method to defeat the only mechanism implemented in the voting machine to protect ballot secrecy. We describe multiple alternatives for correcting the vulnerabilities and discuss realistic scenarios where the confidential aspect of the vote is threatened if the vulnerabilities are not fixed. Chapter 4 presents another collection of flaws detected in the voting software and its development process. Finally, Chapter 5 concludes the document with perspectives on how to improve transparency and auditability of the electronic voting system.

1.4 Acknowledgements

We would like to thank Prof. Pedro Rezende, a colleague at the University of Brasília, and Prof. Jeroen van de Graaf, from Federal University of Minas Gerais, for their useful feedback during the preparation of this work.

[1]From the neologism in Portuguese *zerésima*.

Chapter 2

Public Security Tests

Formally, the event began with the publication of a call for participation and team registration. According to the official document [2], only the teams approved by the SEC would have the opportunity to participate in the trials. The major difference between the 2nd and 1st editions of the trials was access to the source code of the voting software. The 1st edition of the event consisted exclusively of "black box" testing.

2.1 Format

The 9 approved teams composed of 24 investigators participated in two stages spanning 3 days with 10 hours a day of activities: (i) a preparation phase, between March 6th and 8th, 2012, when the teams could study the voting software source code and ask technical questions to formulate hypotheses and testing plans to evaluate the quality of security features implemented in the voting machine; (ii) a testing phase, between March 20th and 22nd, 2012, when teams could no longer study the source code and could exercise their methodologies to validate hypotheses and obtain results and conclusions.

Concrete activities of the 2nd edition of the Public Security Tests started on March 6th, 2012, with an opening talk [3] where the format and rules of the event were presented together with an overview of the voting procedures and security measures implemented in the voting machine. The goal of the opening talk was leveling the amount of information available to the participants. The team composed by the authors, identified as "Group 1", attended the opening talk to familiarize themselves with technical aspects of the system and to detect promising points of attack.

During the 12-day period between the two phases, teams were required to submit testing plans formulated from the information collected during the preparation phase. Only testing plants approved by the Disciplinary Committee of the event (appointed by the SEC) could be put into practice

in the following phase. The restriction on access to source code during the testing phase was waived on the second day of the testing phase. The authors did not take advantage of this possibility.

2.2 Objectives

The call for participation explicitly divided the objectives of the trials into two distinct classes [2]:

- *Failure*: event when a system violates its specification after entering an inconsistent state of execution caused by a fault or imperfection in the software or hardware components, and improper functioning does not have any interference on the destination or anonymity of the votes.
- *Fraud*: intentional act of modifying information or causing damage with impact on the destination or anonymity of the votes, preferably without leaving apparent traces.

The first class is composed by *denial of service* attacks, where an attacker aims only to make the voting equipment unavailable to the electors. The second class captures attempts at electoral fraud.

The team formulated and submitted two testing plans, titled "Untraceable attempt of compromising ballot secrecy" [5] and "Untraceable attempt of corrupting election results" [6], both clearly directed to cause fraud in a simulated election using official procedures. Due to time restrictions, only the first testing plan was put into practice.

2.3 Methodology

The method proposed by the testing plan required the team to split into two parts, here identified by A and B, who alternated their presence in the testing room to avoid any kind of internal communication. The experiments followed the procedures below:

1. Generation by the SEC of a secret list of fictional votes for city councilor and mayor.
2. Receiving of the secret list of votes by part A of the team.
3. Software installation of the voting machine using an official memory card and printing of the *zeroth*.
4. Casting of votes in the voting machine by part A of the team, following the order of the list and under supervision of SEC officials.

5. Violation of the seal and delivery of the Media of Results (MR) to part B of the team.

6. Execution of a customized program to analyze the Digital Record of the Votes (DRV) stored into the MR and produce a list of votes in order which supposedly correspond to the votes cast on the voting machine.

7. Comparison of the list of votes kept secret to part B and the list of votes produced by the customized program.

The success criteria for the method is naturally the correspondence between the two lists. Observe that, inside the testing room, part B of the team had to violate a seal and retrieve the MR to complete the simulation, since this was the only way to obtain the DRV matching the simulated election. In real elections, the DRV is public by law [7]. Part A also needed physical access to the voting machine, but only to cast the prescribed votes, according to the protocol described above.

2.4 Results

As stated in the report jointly written by the authors and the SEC [5], the ballot secrecy attack method obtained absolute success in recovering the votes in the order they were cast during simulated elections with 10, 16, 21 and 475 electors (20, 32, 42 and 950 votes, respectively). The latter reproduced the proof-of-concept results with a realistic amount of data, a requirement made by the SEC to match the 82% participation rate from the previous election in the universe of 580 fictional electors who composed the training set of the event.[1] Since the attack method only consisted on analyzing public products of an election, no modification in any component of the voting machine or invasion of its security perimeter was needed. For this reason, the method is essentially untraceable.

Storing the votes in an order different than the order they were cast is a critical procedure for protecting ballot secrecy. It is clear that the authors' methodology defeated the only security mechanism employed by the voting machine to protect ballot secrecy. It was not possible, however, to recover the ordered list of elector identities from the public products of an election. This information must be obtained externally in order to relate the ordered votes with the ordered identities, making possible an exact correspondence between each voter and its vote. To the extent the authors could investigate, public products only store the missing electors in lexicographic order of registration numbers. The next chapter discusses how recovering the ordered votes allow electoral fraud in realistic scenarios.

[1] Voting in Brazil is mandatory, thus the high participation rate.

There was not sufficient time to execute the second testing plan, which aimed to evaluate the security measures that protect the integrity of results. Priority was given to the first testing plan because of its simplicity and almost complete independence from any significant collaboration with the SEC. Attacking the integrity of results requires the electoral authority to at least attest the authenticity of the corrupted results with the existing detection measures, requiring more time to execute.

2.5 Scoring

Scoring criteria were devised by the SEC to objectively compare the performance of the teams [8]. Without detailed justification and even with the absolute success during the execution of the testing plan, the authors received the negligible score of 0,0313 on a 0-400 scale [9]. The penalties applied to the team score were questionable at best. It was not clear, for example, why penalties caused by the necessity of intervention points only existing on the testing environment where applied even if they were not required on a real instantiation of the attack. The Evaluation Committee of the event (also appointed by the SEC) understood as four the number of intervention points: physical access to the voting machine, protection seal and memory cards and access to the source code. It would be impossible to simulate any election without physical access to the voting machine and it would be impossible to analyze the public products of a simulated election without breaking the seal to retrieve the Media of Results. The attack did not require access to the voting machine beyond what is allowed to electors during the voting process or mandated to election officials at the end of the voting session. Political parties receive the contents of the Media of Results without physical access to the voting machine. Is is also incoherent to penalize the team for visualizing the voting software source code, when the objective of the event was to evaluate the quality of security features implemented in that source code. The team still does not understand why their methodology was considered to be an attempt to cause failure instead of a fraud attempt on a simulated election, since no apparent failure was perceived in the voting equipment during the whole trials. Despite all scoring issues, the team won the competition after providing the most significant contribution to improve the security of the electronic voting system.

There are two possible hypotheses for the negligible team score: either the Evaluation Committee did not understand the severity of the vulnerability exploited or this was a deliberate attempt to mischaracterize and quantitatively minimize the results. Both hypotheses are equally worrisome.

2.6 Improvement

During their participation, the authors collected several recommendations to improve the event:

- **Minimize the intervention from the event staff**: the necessity to monitor the investigators during the execution of their testing plans is comprehensible, but the lack of privacy and constant intervention disrupted the well functioning of the team.
- **Minimize bureaucracy**: again, the necessity of keeping track of all the procedures executed by the investigators is perfectly comprehensible, but satisfying bureaucratic requirements consumed an amount of time which could be dedicated to the execution of additional testing plans.
- **Minimize the time restriction**: 30 hours are absolutely insufficient to analyze a significant portion of the voting machine source code, which has in total a few million lines. Mission critical software should be considered security software in all of its entirety, since a vulnerability in non-critical code can trigger a vulnerability in critical code. Because of this, the duration of the event should be maximized.
- **Amplify the source code availability**: a sealed room with only 4 computers was specifically dedicated to study the source code. This lack of capacity severely reduced the amount of exposure of the source code. In particular, the team only obtained access to the source code at 11 AM of the second day of the preparation phase, since another team obtained exclusive access to the sealed room on the first day. In total, the team spent only 5 hours of the preparation phase studying critical portions of the source code. The availability of simple text processing utilities (`grep`, `vi`, `cat`, etc) is paramount for the efficient detection of which code sections present higher interest.
- **Amplify the testing scope**: the event focused exclusively on the security mechanisms implemented in the voting machine. The SEC provided the reasonable at first justification that any entity can perform a parallel totalization of the results afters all the partial numbers are published on the Internet. This way, any attack directed to the totalizator would only delay the publication of the official results. However, in our opinion, successful attacks directed to the centralized totalizator could create ambiguity or corruption of the official results. These can be detected and neutralized afterwards, but only when the respective guarantees, that the correct results obtained by each voting machine correspond to the ones published in the Internet, are available to any potentially damaged candidates. A successful attack of

this type would still put into question the reputation and capacity of the electoral authority in executing the elections or even what is the correct election outcome.

- **Improve the scoring criteria**: the formula to objectively evaluate the performance of the teams was ill-conceived and had too much focus on applying penalties. The official report written by the Evaluation Committee did not justify their decisions and only listed the intervention points and final scores.

- **Change the nature of the event**: the competition format does not put incentive into information sharing and emphasizes cost-benefit metrics. This restricts the scope of deciding which the vulnerabilities detected will allow the fastest attack method. This way, more sophisticated methodologies lose priority because they require more effort. These characteristics clearly model a portion of potential attackers, but only a careful collaborative evaluation of security mechanisms allow the modeling of well-informed attackers with considerable resources to represent persistent threats.

The complete and careful evaluation of the voting machine software requires enormous amounts of effort and time. Without the possibility of extensive unrestricted testing, following a sound scientific methodology, it can not be said that the current format of the event significantly improves the security of the voting system. It only allows the detection of easily exploitable vulnerabilities which allow simple attacks with limited effects.

Chapter 3

Vulnerabilities

In this chapter we describe the sequence of vulnerabilities which allowed the team of authors to recover the list of ordered votes in several consecutive simulated elections, one of them using a realistic number of electors.

3.1 Digital Record of Votes (DRV)

Since the officialization by electoral law of the current DRE voting machines in 1997 [10], voter-verified paper audit trails (VVPATs) were adopted in Brazilian elections for the first time in 2002 [11]. They aimed to distribute among all electors, agents with the higher interest in a reliable democratic process, the possibility of independent verification of their individual votes. Paper audit trails consist of voter-verified materialized versions of the votes that can be stored for later recount without allowing electors to prove their choices to any interested parties. Without independent verification of results, trust has to be put on the limited software auditing measures exercised by the political parties before the election and on the good faith of the technicians responsible for the voting system. After allegations by the election authority that the additional printers increased cost significantly and created many operational problems, VVPATs were suspended [7]. In their place, a purely digital replacement was adopted.

The DRV is a table separated in sections, where each section is devoted to a different public office position selected by the election. This table shuffles the votes cast by the electors during storage to disassociate the order of the votes and the order of electors. It was introduced as a replacement to VVPATs to supposedly permit independent verification of election results. For this reason, it is a public document made available to the political parties after the elections. However, while paper audit trails in fact allow independent verification of the votes computed electronically, the DRV is produced by the same software component which tallies the votes and produces per-machine partial summations. This way, any successful attack against the

tallying process can also compromise the integrity of the DRV.

Hence, the DRV does not serve any practical purpose besides compromising ballot secrecy if it is not designed or implemented securely. Figure 3.1 presents a fictitious DRV for an election with 3 public office positions and 7 electors of which only 3 participated. The first elector choses candidate number 13 for Governor, 31 for Senator and casts a BLANK vote for President. The second elector chooses 71 for Governor, casts a NULL vote for Senator by inputting an invalid number and chooses 37 for President. The third and last voter also chooses 71 for Governor, casts a BLANK vote for Senator and chooses 37 for President. Observe that the final version of the file apparently does not allow recovery of any correspondence between electors and their votes and that unused positions are conserved by the shuffling process.

Governor	Senator	President
	31	
13		
		BLANK

(a) Storage of first vote.

Governor	Senator	President
71	31	
13		
	NULL	
		BLANK
		37

(b) Storage of second vote.

Governor	Senator	President
71	31	37
	BLANK	
13		
71	NULL	
		BLANK
		37

(c) Storage of third vote.

Governor	Senator	President
71	31	37
	BLANK	
13		
71	NULL	
		BLANK
		37

(d) Final aspect of the file.

Figure 3.1: Example of shuffled storage of votes in the DRV.

3.2 Hypothesis

The shuffling mechanism was presented in the opening talk and immediately raised suspicion among the team. [3]. The reason for this was the clear observation that the shuffling should reach cryptographic strength, and only someone with proper training in computer security would recognize that this is as important for ballot secrecy as software integrity is for reliable tallying. Still during the opening talk, the team raised the hypothesis that the DRV was not designed and implemented securely. With only a few recursive searches for well-known insecure functions for random number generation in the first hour of source code studying, the hypothesis was considerably strengthened. It only remained to determine which data was needed to revert the shuffling and recover the votes in the order they were cast.

3.3 Design and implementation

The shuffling mechanism was designed and implemented with a progression of errors which culminated in allowing its reversal. The implementation uses a pseudo-random number generator, a computational procedure which produces a sequence of numbers apparently random, but that can be uniquely determined from a small parameter called *seed* which must be chosen in a truly random fashion. When the sequence of numbers should be protected from independent derivation by an attacker, the seed must not only be truly random but also be kept in secret. In the following, we present the progression of software vulnerabilities that forced the pseudo-random number generator to work out of its operation limits, not fully reaching its security properties:

1. **Inadequate choice of pseudo-random number generator:** the standard generator included in the C programming language and implemented through functions `rand()`/`srand()` was chosen. This generator has an extremely short period and accepts seeds with only 32 bits. Thus, it does not reach cryptographic strength [4]. Just this choice of generator already allows a probabilistic attack method.

2. **Inadequate choice of seed:** the seed was chosen at the initialization of the voting software as a time measurement with precision of seconds in the UTC timezone and implemented through function `time()`. This choice of seed is obviously not truly random. The system must be initialized on election day between 7 and 8 AM and this information alone reduced the exhaustive search space to just 3600 values.

3. **Public seed:** the seed was not only deterministic but also made public in the LOG of events and in the *zeroth* official document. The former

becomes public to the political parties after the election, while the latter becomes public right after its printing, when it receives handwritten signatures by election officials and inspectors from the political parties. Given the right time that the *zeroth* was printed, it is trivial to recover the ordered votes efficiently and exactly, without any error probability or need for an exhaustive search. The digital signature mechanism on the LOG file and the handwritten signatures on the *zeroth* further guarantee that the documents are authentic and the timestamp contained in them is indeed the correct seed.

Algorithms 3.1 and 3.2 present simplified versions of how the pseudo-random number generator was initialized and how votes were stored in the DRV, respectively. Figure 3.2 presents a copy of a real *zeroth* found on the Internet, with the seed (which should be random and secret) circled in red. Let n be the number of electors who voted in an election with m total electors. The way the DRV conserves the empty positions allows one to try different values for the seed and always obtain the correct one when $n < m$. This test is possible by comparing the empty positions in the DRV with the empty positions generated by storing votes of n electors with the potential seed being tested.

Algorithm 3.1 DRV Initialization.

Input: Table T representing the DRV, total of m electors.
Output: Table T initialized and pseudo-random number generator seeded with a timestamp.

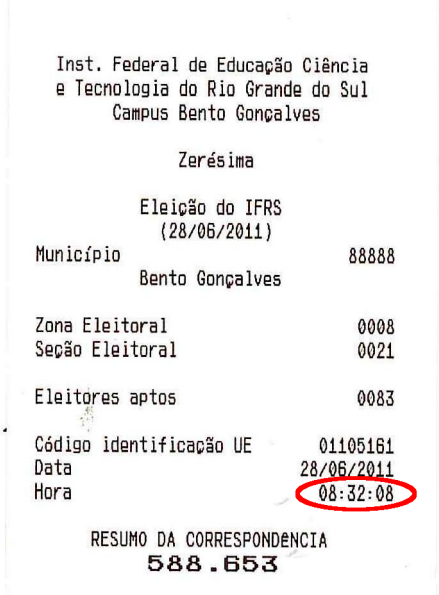
1: `srand(time(NULL));`
2: **for** $i \leftarrow 0$ **to** m **do**
3: $\quad T[i] \leftarrow$ EMPTY
4: **end for**

Algorithm 3.2 Storage of a vote in the DRV.

Input: Table T representing the DRV, i-th vote V, with $0 \leq i < n$.
Output: Table T updated with vote V stored.

1: $j \leftarrow$ `rand()` $\bmod m$
2: **if** $T[j] \neq$ EMPTY **then**
3: $\quad$ {Collision found!}
4: $\quad$ Increment or decrement j until a new free position is found
5: **end if**
6: $T[j] \leftarrow V$

Figure 3.2: Document showing the seed for shuffling votes during storage.



Inst. Federal de Educação Ciência
e Tecnologia do Rio Grande do Sul
Campus Bento Gonçalves

Zerésima

Eleição do IFRS
(28/06/2011)

Município 88888
Bento Gonçalves

Zona Eleitoral 0008
Seção Eleitoral 0021

Eleitores aptos 0083

Código identificação UE 01105161
Data 28/06/2011
Hora 08:32:08

RESUMO DA CORRESPONDÊNCIA
588.653

3.4 Attacks

The progression of vulnerabilities presented in the last section allows the formulation of two attack methodologies:

- **Direct attack:** from the seed, which was not supposed to be public, recovered from the LOG file or *zeroth* corresponding to an electoral section, it is possible to simulate the shuffled storage of n votes and detect in which position of the public DRV each vote was stored, making possible the recovery of all votes in order, only from documents specified by the current system as essential for making the electoral process auditable.
- **Indirect attack:** from the votes stored out of order, it is possible to run an exhaustive search in the seed space and discover what is the correct seed by comparing empty positions. Given the correct seed, the direct attack can be executed.

Both attacks above are essentially untraceable, since they do not involve modification of any software or hardware component of the voting machine and do not require invasion of its physical perimeter. Reading public products of an election never leaves traces, since it is not possible to differentiating between inspection for auditing purposes and attacks on ballot secrecy. The attacks are also deterministic, exact and reproducible with no error probability. It becomes clear that the only mechanism used by the voting

machine software to protect ballot secrecy was defeated. This is aggravated by the fact that secret ballots are a constitutional requirement in Brazil. Algorithm 3.3 presents the direct attack described above. After the trials, the team obtained the information that the public LOG of events produced by the voting machine also stores the timestamp each vote is cast [13]. When the time information is associated with the list of ordered votes, it is also possible to recover a specific vote cast in an specific time instant.

Algorithm 3.3 Recovery of ordered votes from the DRV.

Input: Table T representing the DRV, public seed s, number n of electors who voted among m total electors.

Output: List of ordered votes.

1: `srand(`s`);`
2: **for** $i \leftarrow 0$ **to** n **do**
3: $j \leftarrow$ `rand()` $\bmod m$
4: **if** $T[j] =$ MARK **then**
5: {Collision found!}
6: Increment or decrement j until $T[j] \neq$ MARK
7: **end if**
8: Print vote stored in $T[j]$
9: $T[j] \leftarrow$ MARK
10: **end for**

3.5 Consequences

Now suppose an attacker capable of coercing k electors and monitor their behavior on election day. Recovering the list of ordered votes allows this attacker to obtain *mathematical certainty* in different types of electoral fraud violating ballot secrecy[1]:

- Inserting the coerced electors in the k first positions of the voting queue. This does not seem hard to achieve if the attacker funds transportation of electors and arrives early at the voting places.
- Using a marker vote to indicate the beginning of the block of k coerced voters in the voting queue. If arriving early to the voting place is an issue, the attacker can instruct one elector to vote in a previously determined way (nulling his/her vote with an invalid number, for example), after which the sequence of coerced votes begins.
- Registering the identities and position of all electors in the voting queue or the time they cast their votes. This allows an attacker to

[1] This is historically so common in Brazil that it even has its own name in Portuguese: *voto de cabresto*.

break secrecy of all n electors, event those not coerced by the attacker. Observe that this information can be obtained by collaboration with election officers or inspectors from the political parties.

The time an specific vote was cast determines the position in the voting order that a certain elector cast his/her vote. Examining the corresponding position in the ordered list of votes recovered from the DRV directly reveals the choices made by that elector. This directed attack, besides violating a constitutional requirement [14], can cause significant issues for public personalities (politicians, entrepreneurs, ministers). Note that the place and time they vote is frequently reported by the press on election day. For example, the time and place the president of the SEC voted in the last elections was reported by the Court's internal press [15, 16].

3.6 Corrections

Correcting the progression of vulnerabilities starts with strengthening the pseudo-random number generator which determines the positions votes are stored in the DRV. This improvement can be implemented from the components already available in the voting machine. A secure way to perform this correction is replacing the pseudo-random number generator currently used with a cryptographic pseudo-random number generator. Examples of such generators are documented in standards [17] and implementations can be found in general purpose cryptographic libraries [18]. Proper unpredictable seeds also need to be provided for the improved pseudo-random number generator. This real randomness criteria can be satisfied by using a hardware generator based on a well-studied physical effect. According to the specification of the 2009 voting machines [19], a generator with these features is already available in the hardware security module inside the equipment. The AMD Geode processor mentioned in the specification also has a truly random number generator [20] accessible through file `/dev/hw_random`. For previous models, engineering trade-offs must be attained. A possible solution is obtaining the seed through a blocking read from file `/dev/random` which provides entropy of cryptographic quality from compressed operating system events. This approach has problems involving the predictability of the voting system initialization, which may not provide sufficient entropy for a truly random seed, and the lack of entropy impairing the equipment functionality. The last recommended solution is to relax the cryptographic strength and obtain the seed through a non-blocking read from file `/dev/urandom`. In this case, cryptographic strength is lost, but the quality of the shuffling should be significantly better than the current construction.

It is important to test all of the above suggestions and determine if they satisfy minimal security requirements established for the shuffling mecha-

nism. The authors can not be held responsible in case the suggested solutions maintains the shuffled storage of votes still as vulnerable.

Chapter 4

Flaws

Studying the source code of the voting software did not reveal only the vulnerabilities in the design and implementation in the security mechanism to protect ballot secrecy, as discussed in the previous chapter, but also several flaws in critical software components. Each flaw presented here is a potential vulnerability which allows an internal or external agent to formulate an attack methodology. The presence of flaws in critical software components attests the presence of inherent flaws in the software development process.

4.1 In the software

In the following, several flaws found in the software are discussed, some of them already pointed in the 2002 report prepared by the Brazilian Computer Society (BSC), or previously found in the academic analysis of the voting software used in U.S. elections [21]. Diebold manufactured the hardware for the Brazilian and U.S. voting machines, the software for the U.S. equipment and the voting software for initial versions of the Brazilian model. Nowadays, the SEC is responsible for producing all software running in the Brazilian voting machines.

4.1.1 Inadequate protection of ballot secrecy

The Digital Record of Votes (DRV), introduced by a legal device in 2003 and presented in the previous chapter does not provide any real independent verification of results because it is generated by the same software component which counts votes and produces the Partial Summation (PS). For this reason, the possibility of compromising the PS directly implies the possibility of compromising the DRV. This means that the DRV is just redundant information as fragile as what it tries to protect. Since the DRV does not have any practical value, it serves only as a source of attacks against ballot secrecy if the shuffled storage of votes is not designed and implemented securely.

Besides that, the voting machine project does not completely eliminate the possibility of associating the elector identities and their votes through malicious software [12], since the two terminals which collect this information are electronically connected. The required information exists in the internal state of the voting machine at some point and can be captured by malicious software.

The DRV already has 9 years of history and the question if the vulnerability discussed in the previous chapter was also present in the voting software used in 4 past elections (2004, 2006, 2008 and 2010) poses an interesting possibility. While the authors do not currently have any intention of investigating this issue, there are only three possibilities: (i) the shuffling mechanism used in past elections was more vulnerable than the one examined by the team; (ii) the shuffling mechanism used in past elections was as vulnerable as the one examined by the team; (iii) the shuffling mechanism used in past elections was less vulnerable than the one examined by the team. The first two hypotheses indicate that there was inadequate protection to ballot secrecy in 4 past elections, leaving this security property open to attack by internal or external agents with some knowledge of the mechanism. The third hypothesis indicates that the quality of the voting software decays with time, pointing fundamental problems in how the software is developed. The three possibilities are then equally worrisome, specially when it is considered that secret ballots are required by the Brazilian Constitution and that the country has been a fertile field for electoral fraud based on voter coercion for most of its history.

Recommendation. *Eliminate the DRV and replace it by a mechanism which allows truly independent verification of results such as a voter-verified paper audit trail. If the presence of the DRV is still a requirement, we recommend at least that the empty positions are eliminated from the final version of the file. This makes an exhaustive search in the seed space much harder. If the shuffled storage of votes is still vulnerable, this compression does not resist to insider or well-informed attackers.*

4.1.2 Inadequate entropy source

Entropy has a critical aspect to several cryptographic operations which require random data, such as generation of ephemeral keys or seeding of pseudo-random number generators. In many cases, it is possible to completely circumvent the cryptographic primitive by only attacking its entropy source. Obtaining sufficient entropy in devices with limited interactivity through software-only resources is practically impossible. As discussed in the previous chapter, the voting machine software used only a time measurement with resolution of seconds as entropy source, even when better sources where available in hardware.

Collecting predictable information as an inadequate entropy source is

not an unknown or new vulnerability in either voting systems or commercial software. The voting machine used in the U.S. employed equally insecure techniques [21, Issue 5.2.12], obtaining information from the screen contents and a time measurement with resolution of milliseconds. In 1995, PhD students from University of California, Berkeley, discovered without access to source code that version 1.1 of the Netscape Navigator had the same exact vulnerability [22]. In particular, the seed was obtained using the same function call on line 1 from Algorithm 3.1.

Recommendation. *Adopt the suggestions presented in Section 3.6.*

4.1.3 Insufficient verification of integrity

The Brazilian voting machine has a mechanism for integrity verification of its software as means of detecting if the software was maliciously replaced during its installation or execution. This mechanism varies greatly depending on the presence of a customized hardware security module. Because of this, our analysis will be split into two scenarios.

Voting machines not equipped with a hardware module. Software verification is reduced to itself, being vulnerable to deactivation in case of access to the portions of the software responsible to execute the verification. To reduce this risk, it is common to implement a preliminary integrity check at BIOS (*Basic Input/Output System*) level to guarantee that the software executed next in authentic. However, this technique only reduces the integrity of the software to the integrity of the BIOS firmware. The problem of verifying the BIOS firmware is reduced to itself, without any external source of trust.

Voting machines equipped with a hardware module. BIOS firmware is further checked by the hardware module. In this scenario, the software integrity verification problem is reduced to the authenticity of the source of trust stored inside the hardware module. This can be a self-contained certificate chain to validate digital signatures applied to the other software components. Defeating a software verification mechanism with these characteristics requires collaboration of an insider capable of deactivating the security module or replacing the certificate chain and computing new signatures with the corresponding private keys for the malicious software. However, according to specification of the security module in the 2009 voting machines, the hash values of the BIOS firmware needs to be programmed into the hardware module [19]. This means that the BIOS transmits its own hash value to be verified by the hardware module, instead of requiring that the module actively verify the BIOS firmware. Hence, a malicious BIOS can impersonate the authentic BIOS by transmitting the correct hash values

and deactivate the integrity verification of the software components executed afterwards.

At last, the authors observed that a critical line of code in the application manager responsible for verifying the integrity of shared libraries was deactivated with a comment, confirming that even if a chain of trust is correctly established, software integrity verification is still susceptible to sabotage or programming errors.

The BCS Report already presented an explicitly skeptic position regarding the possibility of software self-verification through cryptographic techniques [12, page 24]. Additionally, guaranteeing that the voting software indeed was produced by the SEC does not make it secure, only confirms its origin, even when the integrity verification mechanism is not circumvented and works correctly.

The software integrity verification problem is endemic in voting systems and is particularly hard to solve in practice. The same limitation in the integrity controls was observed in the voting machines used in the U.S. [21, Issues 4.1.5 and 4.1.6]. For this reason, it is recommended to install means for software-independent auditability of results.

Recommendation. *Perform the verification of the BIOS contents by the hardware security module in an active manner. This recommendation was also suggested by Group 6 participating in the trials [23]. More generally, we recommend transferring the pressure on verifying software integrity to software-independent verification of the results produced by it.*

4.1.4 Sharing of cryptographic keys

Every voting machine in operation uses the same cryptographic key to encrypt the protected partitions of their memory cards. Leakage of this cryptographic key has the devastating impact of revealing to an attacker the entire content of the memory cards, including the voting software, the software integrity verification mechanism and the RSA private key used to digitally sign the public products of an election [24]. The latter is shared by all voting machines in the same state [25] and its leakage allows an attacker to produce a forged file (LOG, DRV, PT) detected as authentic by the centralized totalizator. The hardware security module introduced in newer voting machines also has unused storage capacity for private keys. [19]. We can conclude that confidentiality of the private key and, consequently, integrity of the partial summations depend only on the confidentiality of a cryptographic key shared by half a million machines [3].

In an official position [26], SEC argues that using multiple encryption keys to encrypt the same files can leak statistical characteristics of plain text [26]. Attacks of this nature are indeed studied in cryptographic literature, but do not represent any relevant threat in practice [27]. It is clear that

this risk is nowhere near the consequences of a compromise of the massively shared encryption key. If a proper mode of operation for encryption is used, this risk is trivially eliminated by randomizing the block cipher input when the plain text can not be chosen by the attacker [27], as the case discussed here.

Recommendation. *Assign a different cryptographic key to each voting machine, or at least to each memory card used to install software in a reduced set of voting machines. Key derivation functions are cryptographic tools designed to solve this exact problem.*

4.1.5 Presence of cryptographic keys in the source code

Sharing of cryptographic keys is aggravated by their clear presence in the source code of the voting software. This means that any internal agent with unrestricted access to the versioning repository where source code is kept immediately has access to the cryptographic key which protects the encrypted partitions of all memory cards. This also means that the encryption key is part of the operating system module responsible for mounting the encrypted partitions and making their contents available. Thus, it must be stored in the plain text portion of the memory cards. The encrypted objects are stored right besides the cryptographic keys which decrypt it, qualifying this mechanism as an obfuscation instead of security measure. Leaking the key becomes possible for anyone knowing or able to discover the position in which the key is stored by simply analyzing the plain text portions of the software.

Recommendation. *Store the encryption key in the hardware security module or preferably in a tamper-resistant device external to the voting machine environment.*

4.1.6 Inadequate use of encryption

The encryption algorithm used to protect the encrypted partitions of the memory cards is the Advanced Encryption Standard (AES) [28] at the security level of 256 bits, a recommended choice for critical applications. The selected mode of operation is Cipher Block Chaining (CBC). The combination of algorithm and mode of operation is particularly good. However, the mode of operation uses not only the same encryption key for all voting machines, but also the same initialization vector, element responsible for randomizing the block cipher input and eliminating undesirable leakage of statistical characteristics of the plain text. Choosing a new random initialization vector for each encryption operation is a requirement for this mode of operation [29]. Arguing that using the same encryption key for all voting machines to prevent statistical leakage [26] loses any meaning when the way the mode of operation is used violates its specification.

Recommendation. *Select a new initialization vector for each encryption operation executed by the voting machine software, respecting the original specification of the chosen mode of operation.*

4.1.7 Inadequate choice of algorithms

Algorithms were not only badly chosen for pseudo-random number generator. The voting machine software also employed the SHA-1 [30] hash functions for computing digital signatures and verifying software integrity. This specific hash function is not recommended for such applications since 2006, when it was discovered that it does not offer collision resistance. Rapid migration to secure hash functions was also recommended in that occasion [31].

Recommendation. *Employ a pseudo-random number generator of cryptographic quality, as discussed in Section 3.6; and a collision-resistant cryptographic hash function, for example, from the SHA-2 family [30]. If the length of hash values is crucial for human verification, it is possible to truncate the output of stronger hash functions.*

4.1.8 Repeated implementation of cryptographic primitives

The authors found several repeated implementation of cryptographic algorithms in the code base. Apparently, every software component which employs cryptography in some way receives its own implementation of the involved algorithms, making the proper auditing of all the implementations much harder and significantly increasing the chance of error.

Recommendation. *Concentrate all implementations of cryptography in the same library of critical functionality to ease auditing of their correct functioning. Using a well-known general-purpose cryptographic library such as OpenSSL [18] is also recommended.*

4.2 In the development process

The flaws discussed in the previous Section are the product of a fragile software development process. From now on, we discuss flaws found or inferred by context in this development process. Many of the same problems were also detected in the development process used in the U.S. voting machines manufactured by Diebold [21, Section 4.3].

4.2.1 Complexity

Security is a result of simplicity, transparency and correct evaluation of trust assumptions and conditions. The millions of source code lines required to carry out simple elections in Brazil eliminates any reasonable possibility of a full and effective software audit review. It can be argued that a significant

volume of this software is dedicated to the operating system and thus does not need a review. However, we verified that the SEC insert code sections into the operating system components. For example, the encryption key is directly inserted into the source code of the operating system module responsible for mounting encrypted partitions. It is also worrisome that insufficient compartmentalization and vulnerabilities in non-critical portions of software can create severe vulnerabilities in critical portions which affect security measures.

A volume of source code of this magnitude will, *inevitably*, have vulnerabilities which can be explored. For this reason, the code base needs to be completely oriented around a small set of critical functionalities. The correct and secure functioning of the equipment should rely on this critical set. As a reference value, researchers who evaluated the Diebold voting software in a 60-day interval concluded that the thousands of lines of code dedicated only to the application layer had such complexity that it is not possible to make them secure [21, Issue 4.1.2].

Recommendation. *Reduce code volume by reuse, compartmentalization and refactoring techniques, as exemplified in Section 4.1.8. Avoiding interventions in the external source code and isolating code portions of the operating system from the application layer can facilitate internal software audit reviews.*

4.2.2 Insufficient external software audit

Inspectors from political parties have the guaranteed right to examine the source code of the voting software, but for this they have to sign a Non-Disclosure Agreement (NDA) which prevents them from publicly disclosing any problem observed in the code. This manner, inspectors can not reveal the quality of the voting software or its security measures in detail, while malicious agents are free to attempt electoral fraud. In the way it was established, inspection from political parties is insufficient to improve security of the system. Since inspection from independent investigators is extremely limited, consisting only in a few days of work and under complete monitoring, or more recently, during a period where the immense code base is constantly modified and under inadequate conditions, in practice no effective auditing is done in the software component of the electronic voting system. This problem was also previously raised by the BCS report [12, page 23].

In DRE voting machines without voter-verified paper trails, integrity of results depend only on software integrity. The scenario discussed here looks perfect for untraceable electoral fraud.

Recommendation. *Provide auditing capabilities to any Brazilian citizen, specialist or not, without any legal impediment.*

4.2.3 No static analysis of source code

The vulnerable function family employed for the shuffled storage of votes is detected as fragile by any tool for static analysis of source code. For example, the free tool *Flawfinder* [32], produces the following warning when it examines code containing the function call, such as our implementation of Algorithm 3.3:

> *This function is not sufficiently random for security-related functions such as key and nonce creation. Use a more secure technique for acquiring random values.*

Recommendation. *Adopt sophisticated tools for static code analysis in order to minimize the impact of programming errors capable of creating severe vulnerabilities, respecting good practices for developing mission-critical software.*

4.2.4 Inappropriate attacker model

The security mechanisms in the Brazilian voting machine are designed to only resist attacks from external attacker and ignores the risk of internal attackers. In particular, as it is made clear by the SEC's official position [26], detection of potentially malicious behavior promoted by internal agents is performed by an auditing process also executed by internal agents. The sharing of encryption keys mentioned previously is a perfect example of this phenomenon, since there is enormous emphasis on esoteric statistical attacks mounted by external attackers while the risk of leakage by insiders is completely ignored. Storing this encryption key as plain text in the voting machine memory cards shows that security is not designed to resist well-informed attackers.

Recommendation. *Adopt security mechanisms resistant to external agents and, particularly, internal agents armed with detailed knowledge of such measures.*

4.2.5 No internal security exercises

In a meeting between the authors and the SEC members responsible for designing and producing the voting machines, right after the public audience of the Public Security Tests, we offered a technical talk to illustrate all the problems found in the software and the reasoning which let us detect and explore the vulnerability discussed in Chapter 3. The offer was well received, because it would allow the interested parties to exactly understand "how the attacker mind works", in the words of the SEC members. There was no further concrete invitation for this, but our reading of this meeting indicates that there is no internal team responsible for periodically simulating an attacker and exercising potential attack methodologies.

Recommendation. *Establish, train and direct an internal team of simulated attackers, a recommended practice for mission critical software [21]. Design of security measures needs to be accompanied by simultaneous attempts of defeating them.*

4.2.6 No formal training

The flaws discussed in this chapter, found even in critical security mechanisms, demonstrate clearly that the SEC members responsible for developing voting software do not receive sufficient training to implement secure software. The hypothesis raised by the authors, still on the opening talks, that the vote shuffling mechanism was not designed and implemented securely due to lack of training confirms this observation. The absence of internal simulations to model plausible attackers due to the lack of understanding of how an attacker works also supports our claim, since any well-trained professional in computer security naturally alternates between the roles of security designer and attacker to test the quality of its own work.

Recommendation. *Provide proper training for the development team to consequently improve the quality of delivered software. It is not realistic to expect secure software as the result of a software development team with no formal training in computer security.*

4.2.7 Critical data made available to investigators

The machines dedicated to expose source code in a sealed room during the Public Security Tests apparently came directly from the development team. The reason for this is the availability to all investigators of critical information regarding usernames, passwords and internal network paths to the software versioning servers. An attacker equipped with this information and able to enter the SEC internal network can malicious modify the source code and make the changes effective under the responsibility of an innocent programmer.

Recommendation. *Sanitize equipment made available to external visitors in a way that critical information is not disclosed.*

4.2.8 Ignorance of relevant literature

As discussed in Chapter 3, the vulnerabilities found in the vote shuffling mechanism are well-known for at least 17 years [22]. Several flaws discussed in this report were already described by technical reports evaluating other voting systems [21], or even the one under discussion [12], and represent the opposite of recommended practices and formal specification of cryptographic techniques. Persistence of these issues in a code base with 16 years of history is unjustifiable and clearly shows that the SEC team responsible for the

electronic voting system does not adequately follow the relevant movements in the field of electronic voting.

Recommendation. *Dedicate explicitly part of the development team to study and distribute relevant advances of practical or academic interest in the area of computer security.*

4.2.9 False sense of security

The incessant repetition that the Brazilian voting machine is unconditionally secure and tamper-resistant, event if this constitutes a theoretical impossibility, disturbs the critical sense of the software development team and culminates on the suspension of their self-evaluation mechanisms. The software development process used in the voting machines apparently works under the effect of *suspension of disbelief*, installing a generalized false sense of security. This is not the ideal environment to develop security measures, specially when these need to satisfy mission critical requirements.

Recommendation. *Install a software development process able to stimulate mutual and critical verification of the work being done, with realistic evaluation parameters.*

Chapter 5

Conclusions and perspectives

This report presented a collection of software vulnerabilities in the Brazilian voting machines which allowed the efficient, exact and untraceable recovery of the ordered votes cast electronically. Associating this information to the ordered list of electors, obtained externally, allows a complete violation of ballot anonymity. The public chronological record of events kept by the voting machines also allows recovering a specific vote cast in a given instant of time. The consequences of these vulnerabilities were discussed under a realistic attacker model and corrections were suggested. Several additional flaws in the software and its development process were detected and discussed with concrete recommendations for mitigation. In particular, it was demonstrated how to defeat the only mechanism employed by the voting machine to protect ballot secrecy.

The necessity of installing a continuous and scientifically sound evaluation of the system, performed by independent specialists from industry or academia, becomes evident and should contribute to the improvement of the security measures adopted by the voting equipment.

This collection of flaws and vulnerabilities provides material evidence for hypotheses already raised by the 2002 BCS Report on the voting system [12]. In particular, we can conclude that there was no significant improvement in security in the last 10 years. Inadequate protection of ballot secrecy, the impossibility in practice of performing a full or minimally effective software review and the insufficient or innocuous verification of software integrity are still worrisome. Since these three properties are critical to guarantee the anonymity and correct destination of votes, the authors repeat the conclusions of the aforementioned report and defend the reintroduction of voter-verified paper audit trails to allow simple software-independent verification of results. Paper audit trails distribute the auditing procedure among all electors, who become responsible for verifying that their votes were correctly registered by the voting machine, as long as a tally is done afterwards to check that the electronic and manual vote counts are identical. This tally-

ing process can be performed in a prescribed portion of the votes to reduce the impact on the availability of results. It is important to emphasize that printed votes are only means for independent verification and should not leave the voting place to serve as proof for external parties, as mandated by the corresponding law [33]. Voter-verified paper audit trails were scheduled to return on the 2014 elections, but unfortunately they were suspended by questionable allegations of unconstitutionality.

A movement in this direction would follow the current trend in electronic voting systems. With field tests being executed by the Indian Election Commission, Brazil is now the only country in the whole world to adopt electronic voting systems without independent verification of results. We believe that for this reason, and considering all the security problems discussed in this report, the software used in the Brazilian voting system does not satisfy minimal and plausible security and transparency requirements.

Bibliography

[1] Janino, G. D.; Balcão Filho, A.; Montes Filho, A.; Lima-Marques, M; Dahab, R.: Report from the Multidisplinary Committee appointed by the Superior Electoral Court (in Portuguese), 2009.

[2] Superior Electoral Court. Call for participation Nº 01/2012 (in Portuguese). http://www.justicaeleitoral.jus.br/arquivos/tse-2-edicao-dos-testes-de-seguranca-na-urna-eletronica

[3] Azevedo, R.: Technical Security Aspects of the Electronic Voting System (in Portuguese). Available at http://www.tse.jus.br/hotSites/testes-publicos-de-seguranca/arquivos/material/Apresentacao_aspectos-tecnicos.pdf

[4] Wheeler, D.: Secure Programming for Linux and Unix HOWTO, 2003. Available at http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO.html

[5] Group 1. Testing Plan GP1T1 – Untraceable attempt of compromising ballot secrecy (in Portuguese). Available at http://www.tse.jus.br/hotSites/testes-publicos-de-seguranca/arquivos/G1PT1.pdf

[6] Group 1. Testing Plan GP1T2 – Untraceable attempt of corrupting election results (in Portuguese). Available at http://www.tse.jus.br/hotSites/testes-publicos-de-seguranca/arquivos/G1PT2.pdf

[7] Presidence of Brazil. Law Nº 10,740, of October 1st, 2003 (in Portuguese). Available at http://www.planalto.gov.br/ccivil_03/leis/2003/l10.740.htm

[8] Superior Electoral Court. Scoring criteria established by document Nº 05/2012 (in Portuguese). Available at http://www.tse.jus.br/hotSites/testes-publicos-de-seguranca/arquivos/TSE-edital-5-2012-criterios-de-classificacao.pdf

[9] Evaluation Committee. Evaluation of the Public Security Tests (in Portuguese). Available at http://www.tse.jus.br/hotSites/testes-publicos-de-seguranca/arquivos/RelatorioFinal.pdf

[10] Presidence of Brazil. Law N° 9,504, of September 30th, 1997 (in Portuguese). Available at http://www.planalto.gov.br/ccivil_03/leis/l9504.htm

[11] Presidence of Brazil. Law N° 10,408, of January 10st, 2002 (in Portuguese). Available at http://www.planalto.gov.br/ccivil_03/leis/2002/L10408.htm

[12] van de Graaf, J.; Custódio, R. F.: Electoral Technology and the Voting Machine – Report of the Brazilian Computer Society (in Portuguese). Available at http://www.sbc.org.br/index.php?option=com_jdownloads&Itemid=195&task=view.download&catid=77&cid=107

[13] Superior Electoral Court. File Specification of the Event Log of the 2008 Voting Machine (in Portuguese), Version 2. Available at http://www.tse.gov.br/internet/eleicoes/arquivos/logs2008/EspecificacaoArquivoRegistroLogUrnasEletronicasEleicoes2008.pdf

[14] Presidence of Brazil. Law N° 4,737, of July 15th, 1965 (in Portuguese). Available at http://www.planalto.gov.br/ccivil_03/leis/l4737.htm

[15] News Agency of the Superior Electoral Court. President of the SEC votes in the capital city of Brazil (in Portuguese). Available at http://agencia.tse.jus.br/sadAdmAgencia/noticiaSearch.do?acao=get&id=1336461

[16] Correio Braziliense. President of the SEC, Ricardo Lewandowski, votes in transit at IESB (in Portuguese). Available at http://www.correiobraziliense.com.br/app/noticia/especiais/eleicoes2010/2010/10/03/interna_eleicoes2010,216159/index.shtml

[17] National Institute of Standards and Technology. FIPS 186-1 – Digital Signature Standard (DSS), 1998.

[18] The OpenSSL Project. Available at http://www.openssl.org/

[19] Superior Electoral Court. Acquitistion of 2009 Voting Machines / Basic Project (in Portuguese). http://www.tse.jus.br/transparencia/arquivos/tse-projeto-basico-audiencia-publica-2009

[20] AMD. Design without compromise, 2007. Available at http://www.amd.com/us/Documents/33358e_lx_900_productb.pdf.

[21] Calandrino, J. A.; Fieldman, A. J.; Halderman, J. A.: Wagner, D.; Yu, H.; Zeller, W. P.: Source Code Review of the Diebold Voting System, 2007. Disponível em https://jhalderm.com/pub/papers/diebold-ttbr07.pdf.

[22] Golberg, I.; Wagner, D.: Randomness and the Netscape Browser. Dr. Dobb's Journal, 1996.

[23] Grupo 6. Testing Plan GP6T1 – Security test of the electronic voting system (in Portuguese). Available at http://www.tse.jus.br/hotSites/testes-publicos-de-seguranca/arquivos/G6PT1.pdf

[24] Superior Electoral Court. 2010 Elections – List of hash values (in Portuguese). Available at http://www.tse.jus.br/arquivos/tse-urna-eletronica-modelo-2009-eleicoes-2010-turno-1-e-2-atualizado-em-22-09-2010-991ue09

[25] Superior Electoral Court. OKEY System, 2010 (in Portuguese). Available at http://www.tse.jus.br/arquivos/tse-chaves-das-u.f.s-eleicoes-2010-turno-1-e-2-991okey

[26] Digital Security Column, by Altieres Rohr. Flaw in voting machine faithfully reproduced error from 1995, says professor (in Portuguese). http://g1.globo.com/platb/seguranca-digital/2012/05/28/falha-na-urna-brasileira-reproduzia-fielmente-erro-de-1995-diz-professor/

[27] Hong, J.; Sarkar, P.: New Applications of Time Memory Data Tradeoffs. ASIACRYPT 2005: 353-372

[28] National Institute of Standards and Technology. FIPS 197 – Advanced Encryption Standard, 2001. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[29] National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation, 2001. Available at http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[30] National Institute of Standards and Technology. FIPS 180-2 – Secure Hash Standard (SHS), 2002. Disponível em http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[31] National Institute of Standards and Technology. NIST comments on Cryptanalytic Attacks on SHA-1, 2006. http://csrc.nist.gov/groups/ST/hash/statement.html

[32] Wheeler, D.: *Flawfinder*. Available at http://www.dwheeler.com/flawfinder/

[33] Presidence of Brazil. Law N^{o} 12,034, of September 29th, 2009. Available at http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2009/lei/l12034.htm