Paakat
Revista de tecnología y sociedac

Mecanismo de auditoría para detección de manipulación de votos en sistemas de votación electrónica

Audit Mechanism for Detecting Voting Manipulation in Electronic Voting Systems

http://dx.doi.org/10.18381/Pk.a8n14.325

Víctor Morales Rocha *
Universidad Autónoma de Ciudad Juárez, México

Óscar Ruiz Hernández **
Universidad Autónoma de Ciudad Juárez, México

Luis Felipe Fernández Martínez ***
Universidad Autónoma de Ciudad Juárez, México

Received: Junuary 6, 2018
Accepted: February 19, 2018

RESUMEN

Los sistemas de voto electrónico son propensos a fallas, errores y ataques intencionales. Un riesgo de estos sistemas es la adición o eliminación de votos directamente en el medio de almacenamiento, usualmente una base de datos. Existen diversos mecanismos diseñados para prevenir, detectar o mitigar las amenazas que pudieran atentar contra la integridad de los resultados de una elección que hace uso de medios electrónicos. Este trabajo presenta una revisión de los diversos mecanismos de auditoría para sistemas de voto electrónico. Además, se presenta una propuesta de generación y protección de logs en sistemas de voto electrónico remoto, con el fin de detectar manipulaciones en los votos almacenados.

Palabras clave
Voto electrónico; seguridad en voto electrónico; manipulación de votos; logs; auditoría

ABSTRACT

Electronic voting systems are prone to faults, errors and intentional attacks. One risk is the addition of fake votes (stuffing) that can be carried out by an external or internal entity. There are several audit mechanisms intended to prevent, detect and / or mitigate any threats that could undermine the election results. This paper presents a review and evaluation of the different mechanisms used to conduct an election audit for electronic voting systems. Also, an audit mechanism to detect ballot stuffing or any other manipulation on votes is presented.

Keywords
Electronic voting; electronic voting security; ballot manipulation; logs; audit

INTRODUCTION

An electronic voting system consists in using digital means in some of the phases of an election, especially to cast votes. Even if it is slow, the use of electronic voting systems is gaining acceptance mainly given the advantages such as the speed in vote counting, the accessibility of visually disabled voters or any other type of disabilities, prevention of errors, among others. Notwithstanding these advantages, an electronic voting system also presents considerable challenges, especially where security is concerned which make voters as well as other electoral participants mistrust said system.

For example, Chun-IFan and Wei-ZheSun (2008) as well as Chaum *et al* (2209) have proposed mechanisms allowing voters to verify that their votes have been registered. Estonia, the pioneer country in the use of electronic voting, has been using an electronic voting system to hold their elections sinxce 2005. According to the description made by Mus, Sabir Kiraz, Cenk and Sertaya (2016), Estonia's voting mechanism consists of three main aspects: 1) voter's applications – a voters application VoterApp and a verification application VerifApp -; 2) a central system: 3) Audit and recount processes. The Estonian system through VeriApp allows verifying that the vote has been cast according to the voter's decision, hence, giving the electoral process greater accuracy.

Bernhard *et al*. (2017) allege that even though vote corroboration mechanisms are incorporated, it is difficult to ensure accurate results. In 2016, during the United States presidential elections, approximately three quarters of North-Americans voted by means of voter verifiable paper ballots (VVPAT, Voter-Verified Paper Audit Trail); however, there was no compelling evidence that the results were accurate or tamper-proof.

On the other hand, should a hacker have access to the database where the votes are stored, he could add illegitimate votes without the individual verification mechanisms being able to detect them. For example, this occurred in the 2012 Florida State elections, where 2,500 illegitimate votes were added in record time.

According to Alcaraz *et al*. (2012), the hacking was discovered thanks to the system that detected abnormalities in the traffic patterns since the demands were coming from a range of IP directions from England, Ireland and India. Should the hacking not have been detected, the number of false votes would have sufficed to affect the election outcome. The detection of this type of hacking is not accessible to the voter, i.e., even though every voter can verify if his vote has been registered correctly, he cannot detect the addition of illegitimate votes.

Therefore, it is necessary to implement audit mechanisms that facilitate the detection of fraudulent practices in electronic voting systems that go beyond individual verification. The following sections describe and analyze the different audit techniques and mechanisms used to conclude the election process which are designed to detect practices that could alter the election outcome; special emphasis is made on the *logging* protection mechanisms. An audit proposal for the detection of vote tampering in electronic voting systems will be described subsequently.

Post-election audit mechanisms

An audit conducted after an election aims at verifying or validating the outcome of the election, as well as to detect if there was some type of error or tampering that could alter said outcome. This section describes and analyzes several audit mechanisms.

Total Vote Recount

It is not surprising that the count of paper ballots in the traditional voting system presents errors since votes are counted by humans. Hence, in case of a total recount, it is highly probable to see a difference between the original count and the recount. Since in the electronic voting system the counting of votes is automatic, a total recount would throw the same results as the original count, assuming that its programming does not present any flaws. Therefore, on the one hand, a total vote recount of paper ballots is far from being reliable, and on the other, a total recount in the electronic voting system will not offer any clue of possible tampering prior to the count.

Total Recount through Independent Means

If we focus on on-site electronic voting systems, an audit may be conducted through an independent verification method[i] as those described in the *Voluntary Voting Systems Guidelines* (VVSG, 2005), which can be used for individual verification,[ii] as well as to carry out an audit. These independent verification systems aim at storing the votes in real time in an alternative means which will be used as backup. Therefore, it is plausible to make a total recount of the votes registered.

However, a recount through an independent means presents a considerable disadvantage. If the recount throws results different from those of the original count, the problem resides in determining which of the two results is more reliable; any of these results could have been tampered with. Therefore, this type of the audit systems presents a challenge in choosing the registration (generated by the voting system or the independent registration); which will be considered valid in case of differences, beyond technology, it is perhaps a topic of electoral legislation that anticipates such situations.

Recount of a Vote Sample

Partial vote recounts represent an audit method frequently used in paper ballot electoral systems. Polling stations are used as units subject to audit or a bigger electoral unit that could be a precinct. The minimal unit subject to audit in in-site electronic voting systems is usually the voting machine or, in certain cases where the system prints a paper backup, the minimal audit unit may be the polling stating where several voting machines are located.

In most countries or states, the decision of carrying out a total or partial recount will depend of the legislation or regulation of the electoral procedures as well as the specific situation that would require an audit; for example, a reason for an audit would be the slight difference in votes in favor of a candidate or another, or an objection process from one or more of the participating parties or candidates. In some states of the United States of America, conducting a partial recount is a standard procedure after an election. For example, in Colorado 5% of the voting machines used must be audited; In Maryland, at least 10% of the polling stations must go through an audit. More examples of parameters and audit procedures can be found under Verified Voting (2017).

The advantages of a partial recount in comparison to a total one are obvious. On the one hand, the cost of resources and time to conduct the audit is significantly less, and on the other, the downsides of total recount are avoided, as explained above. Through a partial recount in which the parameters to determine the sample to be audited are chosen adequately, any tampering can be identified accurately.

The works of Stanislevic (2006), Rivest (2006), Stark (2010), Bretschneider *et al*. (2012), Stark (2015) and Rivest (2017), among others, use statistic and probabilistic models to determine a reliable sample. These models determine the percentage or number of votes, voting machines or polling stations or any other specific unit which must be audited in order to determine with a high level of reliability, if there has been any tampering that might have an impact on the outcome of the election. Some variables are generally taken into consideration such as the size of the election, the vote difference among candidates, the size of the census in the different polling stations or voting machines, among others.

The partial or total vote recounts will only detect the tampering or errors made in the initial count. However, with this type of audits, it is difficult to detect the tampering of altered or eliminated votes as such (whether paper ballots or digital records). On the other hand, partial recounts are not feasible in remote electronic voting systems, i.e., those that use the Internet as means of transmission of the votes since the vote registration is made in a centralized database or, in some cases, in several databases distributed in different physical points; there is not a great number of units to audit. Besides, as mentioned above, a recount through this automated means such as in the case of remote electronic voting system will throw the same result as the original count.

Logging Protection and Verification Systems

Log auditing is a mechanism used to conduct audits in in-site as well as remote electronic voting systems. A *log* is a record of an event that occurred in a digital setting by way of a registry that associates an event with a date and time, user or process that generated it among, other features intended to be registered. Operative systems generally have a *log* registry, i. e., a registry of all the events that occurred in the computer equipment. Furthermore, information systems or applications that have essential functions such as the access to databases, information updating, etc., usually have a registry of specific events of that system for auditing purposes.

It is of great importance that electronic voting systems have a registry of events that gives evidence of the actions carried out more specifically in the cases where tampering is suspected. In a voting session for example, a registry of the voter's authentication, of the event of the choice of the candidate(s), the close of the session, etc., may be generated. Likewise, it is important to have the registry of event of the database, i.e., every transaction carried out directly in the database without having to use the information system.

Notwithstanding the advantages that logs generation offers, it does not cover completely the need to have a reliable audit mechanism since a hacker that has tampered with an election, assuming that he is a user with access privileges, could also tamper with the *logs* to eliminate any evidence of tampering. Technical proposals that allow detecting, up to a certain degree, the tampering made to the *logs* of a system. Next, we describe the most relevant works for the protection of *logs*.

Bellare and Yee (1997) in their paper present a mechanism of *log* protection against tampering. The main concept of their work is that in spite the fact that *logs* can be accessed by any user or external hacker, this user or hacker cannot make any modification without the possibility of leaving a trace. On the other hand, should the contents be eliminated, the mechanism would allow detecting said event and would protect the *logs* through MAC functions or message authentication codes which use private codes.

This security mechanism is based on the fact that it is computationally unfeasible that a hacker alters a message without knowing the private code and yet obtains the same value after applying the MAC functions. In order to avoid that a hacker that has access to a private code tampers with the *logs* generated up to that moment at any given time, the private code may change through time. The private code *ki*, at a given time *ti*, is obtain by applying a hash function to the *ki*-1 code that previously belonged to *ti*-1. Once the *ti* time begins, the *ki*-1code is eliminated. Therefore, if the hacker obtains a private *ki* code, he will not be able to know or deduce a private *ki* code for $j<i$. Hence, the *logs* generated before compromising the private code cannot be modified without being detected.

This mechanism main drawback is the administration of private codes which becomes more complex as the number of codes increases. In order to verify the integrity of the *logs*, all the private codes generated to protect said *logs* must be used. Besides, only the attacks against the modification of *logs* are being considered but not their elimination.

Schneier and Kelsey (1998) propose as another protection mechanism that the *logs* be backed up after a certain time or when a certain number of registries have been obtained. This mechanism considers three components; an unsecured machine, a secured machine and a verifier. *Logs* are generated in an unsecured machine and their backup is performed in a secured machine. It is assumed that there is a connection between the two machines through a data network. A combination of hash functions, digital signatures and cryptography of public and private code are used. As with the mechanism previously described, this mechanism focuses on the protection of *logs* which are generated before being compromised.

Riera and Puiggali (2004) and for Sandler, Derr, Crosby and Wallach (2008) describe other proposals of *logs* protection oriented mainly to electronic voting systems. Riera and Puiggali describe a protocol of *logs* protection for remote electronic voting systems. The idea consists in calculating a hash value of every *log* generated in real time. Furthermore, for each number of *logs* accumulated, or for each specific period a digital signature is applied, then *logs* are chained as follows:

$$\ldots$$

$$Li = H\,(li \,|\, Li\text{-}1)$$

$$Li{+}1 = H\,(li{+}1 \,|\, Li)$$

$$Li{+}2 = H\,(li{+}2 \,|\, Li{+}1)$$

$$Li{+}3 = H\,(li{+}3 \,|\, Li{+}2)$$

$$\ldots$$

Where *l* is a *log* and *L* is the result of the *hash* value of the chaining of *l* with the previous *L*. Every time a *bi* block is formed, a digital signature of said block is carried out:

$$Sigi = [H\,(bi \,|\, sigi\text{-}1)]Sk$$

Where *Sk* is the private code of the computer equipment or server that signs the *logs*. The corresponding public code is used to verify the integrity of said *logs*. The object of establishing an integrity chain between *logs* is to detect if a hacker modifies the *logs*, he will be detected when performing the verification of signatures. Likewise, the elimination of a *log* will be detected when performing the verification of the block to which it pertains.

It is important to highlight that when the quantity of events is very large, it is extremely complex to not only conduct an analysis of the *logs* registries but it become difficult to detect tampering. Conducting a *log* analysis requires tools that automate said analysis, especially when there are large quantities of registries.

Sandler, Derr, Crosby and Wallach (2008) describe a tool called *Querifier* that carries out the *log* analysis in real time which reduces the complexity of analyzing a large quantity of registries. This tool uses definite rules with predicate logic that define all the possible events within a system and the most logical order in which said events must occur. For example, in an electronic voting system, a rule

could be that the “registered vote” event should be preceded by the “authenticated voter” event, in such a way that if the rule is not met, then it would detect some corruption of the registries of events. It also considers if a chain of integrity performed through *hash* functions has been broken, hence, knowing if the tampering of a *log* has occurred.

After conducting an analysis of the mechanisms previously described to conduct audits in remote electronic voting systems, it was noted that these schemes base the detection of tampering mainly on the analysis of the *logs* generated. According to the schemes

described, the difficulty lies not only on the protection of *logs*, but rather on the verification of the integrity, which is a task performed by the auditors, that, without the help of *log* analysis, would be impossible to perform.

Morales (2009)'proposal describes a specific mechanism to detect the adding of votes (attack known in literature as "*stuffing*") by an internal hacker, i.e., a hacker with access privileges to elements of the election or at least to the database where votes are stored. Said proposal describes a technique for the protection of stored votes which allows conducting simpler audits in comparison to the *logs* analysis.

This technique allows the generation of blocks or lots of votes that are backed up in real time. As soon as these blocks of lots of votes are received in a server, the size of every lot is defined, as well as the amount of votes that will form the lot. After completion of the lot, the votes are chained and signed digitally by an authority or several authorities that possess a Sk code as follows:

$$L = \{V1 \mid V2 \mid V3 \mid \ldots \mid Vn\}Sk$$

Every lot of votes is stored in a backup server or in some means of removable storage to be kept in a secure location. Tampering of any of the lots can be detected through digital signature. Moreover, in order to offset a possible hacking through which illegitimate lots could be added and including signing them should the hacker have a private code, an integrity chain of lots is formed as soon as these are generated. To form the chain of integrity, a *hash* value of the previous lot chain is calculated already signed with the current lot and also signed as follows:

$$L'1 = H\,[L1]$$

$$L'2 = H\,[L'1 \mid L2]$$

$$\ldots$$

$$L'n = H\,[L'n\text{-}1 \mid Ln]$$

A sole vote identifier is necessary for auditing purposes to tie the votes contained in the lots with the votes stored in the database. The generation of the sole vote identifier occurs during a vote authentication session. Should illegitimate votes be added in the database, which would be done without a valid voter session. These votes would not have a valid identifier. To conduct an audit, a series of validations comparing the votes included in the vote count with the votes contained in the backup lots must be made. Figure 1 shows the scheme in a general manner:

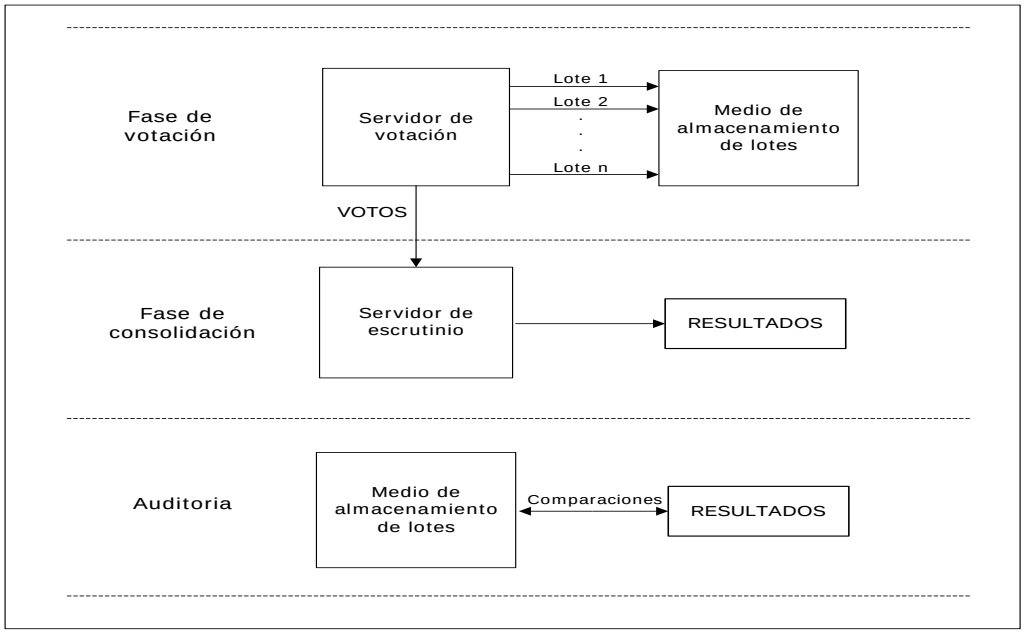

Figure 1. Audit Scheme.
Source: Morales, 2009.

The validations performed in the audit are as follows:

- Verification of the lots integrity. The lots integrity chain and the digital signature of each one of them are revised.

- Verification of the number of votes. A comparison between the quantity of votes included in the vote count and the number of votes contained in the backup lots is made. If the sum of the votes included in the vote count is greater than the amount of votes contained in the lots, it is then possible to know how many illegitimate votes have been added, even though the following step has to be performed to detect which of the votes are illegitimate.

- Verification of the vote identifier. Each vote included in the count should also be found in one of the lots. This validation is obtained by verifying the sole vote identifiers. The votes in a count which identifier does not correspond to the votes registered in the backup lots may be classified as illegitimate votes.

After these validations have been performed in the auditing process, the adding of votes or vote tampering stored in the original database may be verified. This mechanism also can detect if the votes have been tampered with.

Auditing Proposal to Detect Votes Tampering: Medusa

Based on Morales (2009)'s description which was analyzed in the previous section, a mechanism that incorporates some improvements for the detection of the addition of illegitimate votes and the elimination of illegitimate votes has been proposed. This mechanism is called Medusa and, in general, the improvements are: 1) the use of hash to form the chain of integrity instead of the digital signature; and 2) the identification of the last lot to corroborate its existence and integrity. The Medusa operation will be described in detail in the following paragraphs.

As with the previous proposal, backup vote lots are being generated as they are received by the voting server. A vote lot is made up of a certain quantity of votes received that have been linked. A *hash* function is applied and they are signed by an authority or several authorities with the Sk code, as shown here below:

$$L = \{V1 \mid V2 \mid V3 \mid \ldots \mid Vn\} \mid (H\{V1 \mid V2 \mid V3 \mid \ldots \mid Vn\})Sk$$

For example, if it was determined that the lots would contain 100 votes, they would be formed as follows:

$$L1: \{V1, \ldots, V100\} \mid (H\{V1, \ldots, V100\})Sk$$

$$L2: \{V101, \ldots, V200\} \mid (H\{V101, \ldots, V200\})Sk$$

$$\ldots$$

$$Ln: \{V100n\text{-}99, \ldots, V100n\} \mid (H\{V100n\text{-}99, \ldots, V100n\})Sk$$

The signed vote lots are stored in a server or a means other than a voting server. The implementation of a digital signature ensures that any tampering of the lots would be detected, notwithstanding the possibility that if the hacker has access to a private code he may add or eliminate lots without being detected. If the hacking occurs during an audit, the votes counted originally will not correspond to the votes backed-up in the lots.

To prevent these hackings, an integrity chain is formed between the votes and the lots as they are being generated. This integrity chain is made by calculating a hash of the linking of the previous lot signed with the first vote of the current lot, and between the votes by calculating a *hash* to the linking of the previous vote with the current vote.

$$\ldots$$

$$Vi$$

$$V'i = H(Vi \mid Li\text{-}1)$$

$$Vi{+}1$$

$$V'i{+}1 = H(Vi{+}1 \mid Vi)$$

$$Vi{+}2$$

$$V'i+2 = H(Vi+2 \mid Vi+1)$$

$$Vi+3$$

$$V'i+3 = H(Vi+3 \mid Vi+2)$$

…

Where:

- Li-1 is the lot containing votes, *hashes* and signature corresponding to the previous lot.
- V is an individual *log*.
- V' is the result of calculating a *hash* value to the chain of the current vote with the previous vote or lot, i.e., Vi with Vi-1.

As mentioned above, one of the improvements included in Medusa is that the integrity chain does not need to use the linking of the signature to be formed. On the other hand, it uses the calculation of a *hash* of the chain of the first vote and of the previous lot. When implementing the foregoing, the following advantages are obtained:

- Less computer power is required to create and verify the integrity chain since a *hash* function is applied to the link instead of applying a digital signature that requires more processing.
- It is not necessary to calculate the digital signature of every lot to verify its integrity, since it is going to be calculated in those lots where the integrity chain is broken. For example, if we have L1, L2, L3, L4 and L5, and the integrity chain breaks at L3, the integrity can be verified through L2 and L3 digital signature to try to see if there was any modification of if any lot was eliminated.

Likewise, this mechanism proposal considers the possibility of detecting when the last lot or lots were eliminated. For example, if we have five lots L1, L2, L3, L4 and L5, and L5 is eliminated, the integrity chain does not break and the elimination cannot be detected. Another case would be to want to eliminate L3 without leaving any trace; hence, we simply eliminate L3, L4 and L5, and the elimination cannot be detected.

A proposal in trying to mitigate this risk is to identify the last lot, thus offering the possibility to detect if the lot was eliminated or not. To identify the last lost, it has been proposed to make an extra signature, i.e., besides the one made to ensure the integrity of the lot, an additional signature is made to verify that it is the last lot. That is, the chain of the penultimate lot is signed with the last in the following manner:

$$Sig\ n = \{H(Ln-1 \mid Ln)\}Sk$$

Where Sk is the private code of an authority or several authorities of the election in charge of signing the lots digitally. The corresponding public code will be used to verify the integrity and ensure that the last lot exists where it has been stored. It should also be

ensured that only the last lot is marked so that when it is identified, it is necessary to remove the mark of the penultimate lot in order to avoid any confusion.

As with the mechanism proposed by Morales (2009), when conducting an audit, it is necessary that the votes contain a sole identifier to establish a relation between the votes contained in the lots and the votes stores in the voting server, and thus, being able to compare them in an audit. The vote identifier is generated in the voting phase and may be composed of a set of alphanumeric characters.

The sole identifier may also be used to integrate the backup lots according to the electoral unit to which they belong, for example, a municipality, electoral district, state, etc. For example, if an electoral district is assigned identifier "001", the votes issued by the voters of said district may be identified as "001" besides another set of unique characters. This is used to detect tampering directed to a specific geographic area, in order to benefit a specific candidate.

As with the previous mechanism, the generation of a vote identifier can be carried out only until the voter has been authenticated during a voting session. When implementing the foregoing, it is sure that if any invalid votes were added to the database, they would not have a valid identifier. Likewise, if votes were eliminated, it would be possible to identify which ones were eliminated. In both cases, the votes may be identified by introducing a cross between the backup lots and the database.

Consideration must be given to the fact that a hacker can add illegitimate votes in three phases of the election process: before starting the voting process, during the voting phase or, once it is over. On the other hand, the elimination of legitimate votes can only occur in two periods: in the voting phase and once that phase has ended. The addition of votes before starting the voting process can be detected relatively easily, by means of verifications. For example, before any voting starts, it should be verified if the database where the votes will be stored is blank. The addition and elimination of votes in the voting phase itself and even when it has ended can be detected through the proposed audit mechanism as it can be seen below.

Should an audit of the voting results needed be carried out once the vote counting is completed, then, some validations can be made by comparing the votes included in the original count with the votes stored in the backup lots, which are protected against tampering. The audit makes use of the proposed mechanism, and is carried out as follows:

1) As for the previous mechanism, the integrity of the lots is verified. Figure 2 shows an example of the validations carried out to detect if a vote has been modified. The chain of integrity of the lots and the digital signature where the chain has broken are verified in order to determine:

    • If the first lot or a set of the first lots were eliminated: if the integrity chain is broken in the first lot and if by means of the digital signature it has been verified that the first lot has not been modified, then, it can be determined that a lot or a set of starting lots has been eliminated.

- If the last lot or a set of the final lots were eliminated: since the last lot was identified, then, the digital signature can be calculated to corroborate that said last lot exists and has not been modified.

- If a lot or set of intermediate lots were eliminated: if it has been verified by means of the digital signature that the previous and current lots did not undergo any modification at the breaking point of the integrity chain, then, it can be determined that some lot or set of lots were eliminated.

- If a vote was modified or eliminated: if it has been verified by means of the digital signature that the chain of integrity was broken and a lot was modified, then, it can be determined that a vote was modified or eliminated by calculating the *hash* of the link of the current vote with the previous vote.

- It can be specified in which lots the chain of integrity was broken: it can be verified if the chain of integrity is preserved or broken by calculating the *hash* of the link of the first vote with the previous lot and thus, determine at which point or points it breaks.

- It can be specified in which votes the chain of integrity of each lot is broken: it can be verified if the chain of integrity is preserved or broken by calculating the *hash* of the link of the current vote with the previous vote and thus, determine at what point or points it breaks.

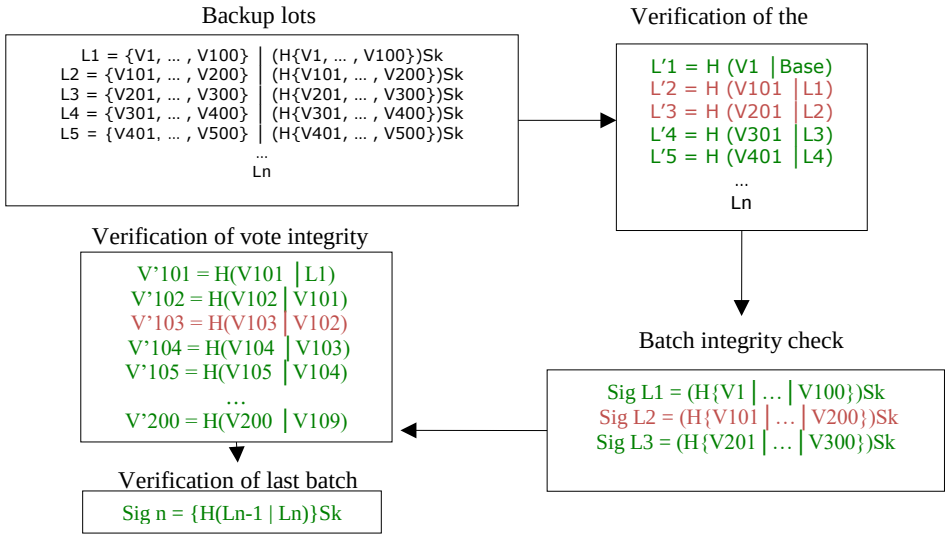

Figure 2. Example of Integrity Verification.
Source: Developed by the author.

2) The number of votes obtained from the count is compared with the number of votes in the backup lots. If the number of votes in the count is different from the votes contained in the lots, it can be deduced that illegitimate votes have been added, or legitimate votes have been eliminated. To know which votes were added or eliminated, the next step must be followed.

3) Since all votes have an identifier, it can be verified that each vote included in the count is also in one of the lots. Votes that are not in the backup lots are classified as illegitimate votes. Counting votes, whose identifiers are not to be found in any of the backup lots are classified as deleted votes.

Through these validations you can obtain evidence of whether illegitimate votes have been added or legitimate votes have been eliminated from the original database, and even such votes can be identified as false. If necessary, a new count of votes can be carried out.

An audit can throw evidence that may suggest that tampering has occurred, and that it possibly requires a deeper analysis than that provided by the proposed mechanism. If that were the case, then some mechanisms allowing *logs* analysis may be implemented, as explained above.

Medusa Prototype

The proposed mechanism is focused on the detection of addition and/or elimination of illegitimate votes; however, it can also protect from and detect the tampering of the *logs* that generated some information systems that handle sensitive information. The National Laboratory of Information Technologies of the Autonomous University of Ciudad Juárez developed a prototype based on the Medusa mechanism for the generation and protection of *logs*, as well as the verification of the integrity of the latter.

The prototype was part of the ERRS project (*Elections Results Reliable Sampling*), aiming at the verification of electoral results. This prototype was developed in *JavaScript* and used *Node.js* as the execution setting. The prototype was performed as follows:

1) When generating a new *log*, a text file is also generated for each minute of the system operation, for example "201708170212.txt", where the *logs* will be stored -see Figure 3-. When generating a *log*, a *hash* value of the link of the *log* generated with the previous *log* is calculated in real time, this with the object of generating an integrity chain between the *logs*; the *log* files have the structure shown in the Figure 4:

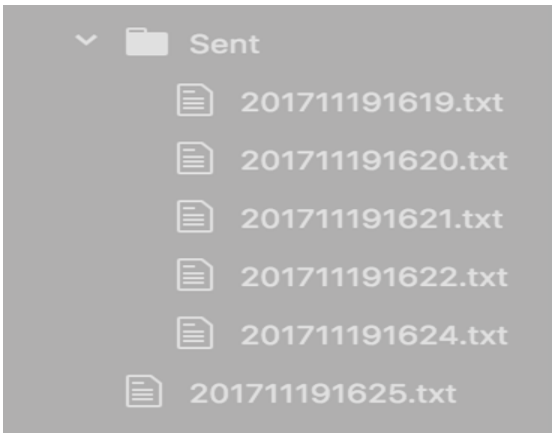

Figure 3. *Log* files.
Source: Developed by the author.

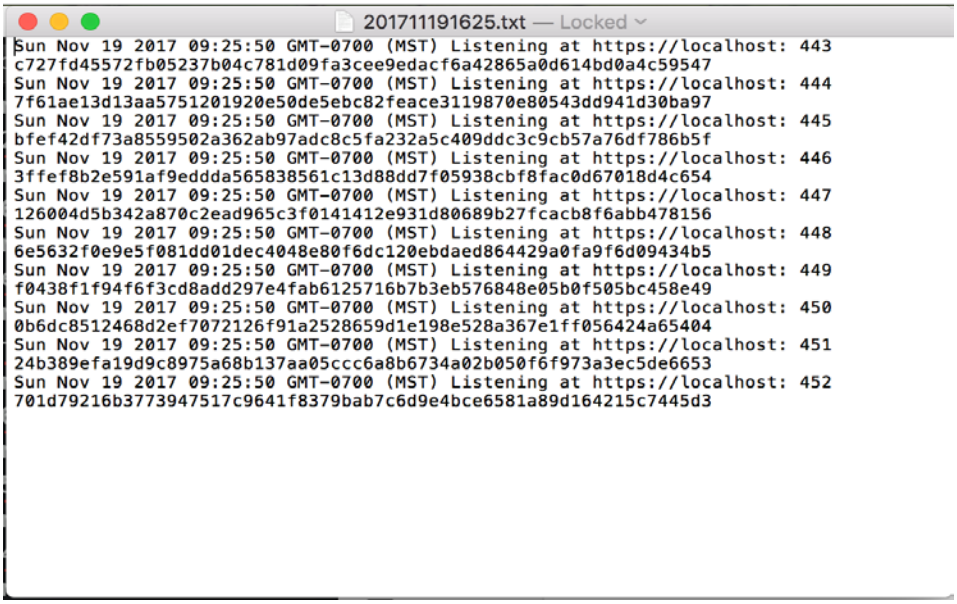

Figure 4. Files Structure.
Source: Developed by the author.

2) When a new file is created, the previous file is signed using a digital signature algorithm to ensure the integrity of these during storage and path, as they are sent to an external server for backup and assurance. In addition to ensuring the integrity of each file also between the files, an integrity chain is formed by calculating a *hash* of the concatenation of the first *log* with the previous *log* file.

The chain of integrity is implemented with the purpose of detecting the breaking point, that is, where modifications were presented. In addition to the signature to protect the integrity of the file, a second signature is applied to mark the file as the last one. The structure of a signed file is shown in figure 5.

3) Once the file is signed it is sent to another server for backup. The backup server is always waiting to receive the *log* files. Upon receiving them, it verifies if the file has not undergone any modification during its journey. This validation is done through the verification of the digital signature, see figure 6.

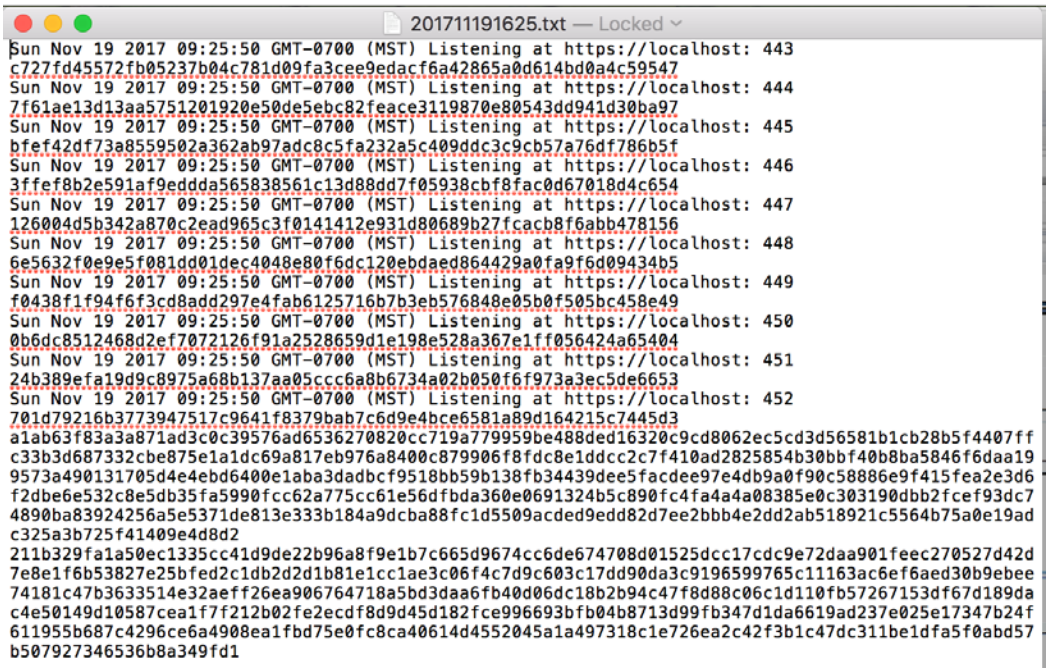

Figure 5. Signed File.
Source: Developed by the author.

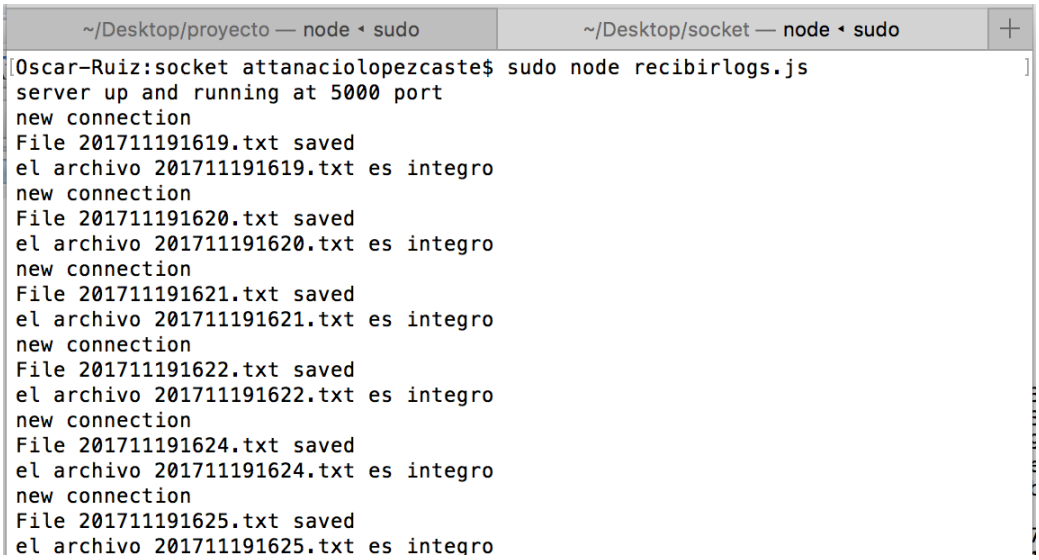

Figure 6. Backup Server.
Source: Developed by the author.

4) Once the *log* files are received, their integrity can be verified at any time. The verification consists in verifying the chain of integrity that is formed between each *log* file, and if the chain is preserved, a message is shown indicating that the files have not been modified and that the chain of integrity is preserved, see Figure 7. If the chain has a breakpoint, the integrity of the files is then verified at the breaking point of the chain to inquire if there was any modification or if a *log* or a *log* file was deleted. In any of these cases, a message is displayed indicating in which file the chain of integrity was broken and, if possible, the reason of the break; see Figure 8.

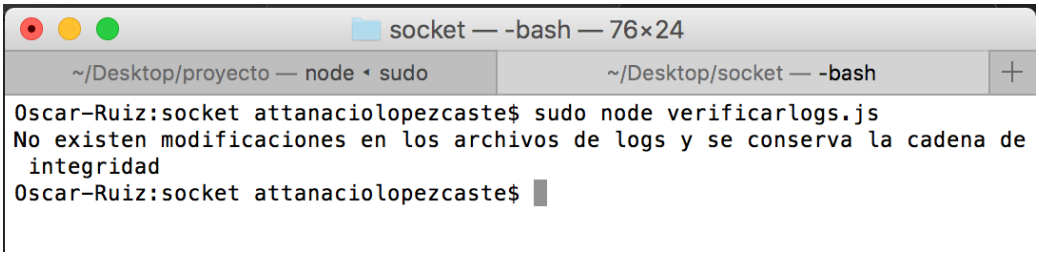

Figure 7. Successful Verification.
Source: Developed by the author.

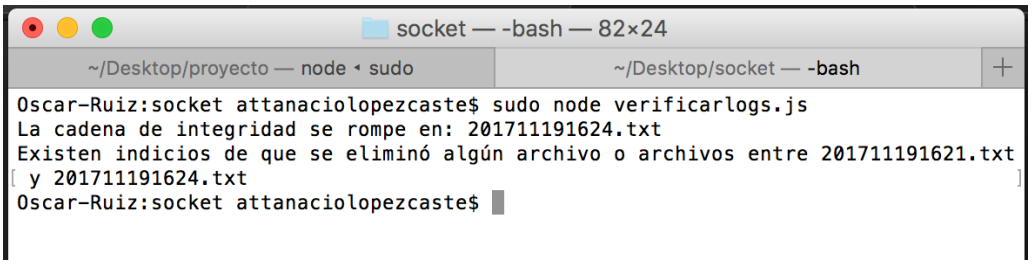

Figure 8. Failed Verification.
Source: Developed by the author.

As explained earlier, the prototype of the Medusa mechanism has a series of verifications that allows discovering *log* tampering. As seen in the implementation example, in the same way *logs* can be protected, so can larger units of digital information such as votes or any other sensitive information.

Conclusions

In this paper we have analyzed the methods, techniques and auditing mechanisms that can be applied to the different voting systems, starting with the classic methods of total and partial recounts by choosing a sample. It has also been described how independent verification systems applied to electronic voting facilitate auditing; however, this is only for on-site electronic voting systems.

On the other hand, there are more adequate techniques to conduct audits of remote electronic voting systems. Some of them are based on the analysis of *logs* or records of events generated during the voting process. However, the management and analysis of *logs* presents some important challenges given the large number of *logs* that can be generated in a system. Automatic *log* analysis tools, such as the *querifier*, facilitate the auditor's task in detecting possible tampering.

This paper describes a mechanism that allows detecting, by means of an audit, votes that were added, eliminated or tampered with in an illegitimate manner. Through this mechanism it is possible to detect with great accuracy where along the *log* chain did the tampering occurred. This gives evidence not only that tampering occurred, but it also deduces in what sense the tampering detected has occurred. Therefore, this proposal contributes to the auditing of remote electronic voting systems; however, the same mechanism can be applied to other types of systems as shown in the example of the implementation of the Medusa mechanism where the *logs* of a system of information of electoral outcome are protected.

Recognition

The authors acknowledge the support of CONACYT, as this paper is supported in part by the following projects: The National Laboratory of Information Technologies - LANTI UACJ headquarters, and Elections Results Reliable Sample (ERRS).

REFERENCES

Alcaraz, J., Aragon, A., Brienen, R., Fiallo, L., Friguls, A. & Gomez, J. (2012). *Final report of the Miami-Dade County grand jury: Report 2012.* Miami Dade County grand jury, Miami, USA.

Adida, B. (2006). Helios: Web-based Open-Audit Voting. *17th USENIX Security Simposium.* Vancouver, Canada.

Bellare, M. & Yee, B. (1997). *Forward integrity for secure audit logs.* University of California at San Diego, Dept. of Computer Science & Engineering.

Bernhard, M., Benaloh, J., Halderman, J. A., Rivest, R. L., A. Ryan, P. Y., Stark, P. B., . . . Wallach, D. S. (2017). Public Evidence from Secret Ballots. *2nd Joint International Conference on Electronic Voting E-VOTE-ID 2017*, (p. 419). Lochau/Bregenz.

Bretschneider, J., Flaherty, S., Goodman, S., Halvorson, M., Jinston, R. & Linderman, M. (2012). *Risk-Limiting Post-Election Audits: Why and How.* California, USA: Risk-Limiting Audits Working Group.

Chaum, D., Essex, A., Carback, R., Clark, J., Popuveniuc, S., Sherman, A. & Vora, P. (Mayo de 2009). Scantegrity: End-to-End Voter-Verifiable Optical- Scan Voting. *IEEE Transactions on Information Forensics and Security , 4* (4), 611-627.

Chun-IFan & Wei-ZheSun (November 2008). An efficient multi-receipt mechanism for uncoercible anonymous electronic voting. *Mathematical and Computer Modelling, 48*, 1611-1627.

Morales, V. (2009). *Seguridad en los Procesos de Voto Electrónico Remoto.* Barcelona, España: Universidad Politecnica de Cataluña.

Mus, K., Sabir Kiraz, M., Cenk, M. & Sertkaya, I. (December 2016). Estonian Voting Verification Mechanism Revisited. *CoRR, abs/1612.00668*.

Riera, A. & Puiggali, J. (2004). *Pnyx Software Requirements Document.* Scytl.

Rivest, R. (November 2006). *Audit On Estimating the Size of a Statistical.* Recuperado de MIT Computer Science & Artificial Intelligence Lab: http://people.csail.mit.edu/rivest/Rivest-OnEstimatingTheSizeOfAStatisticalAudit.pdf

Rivest, R. (January 2017). *ClipAudit: A Simple Risk-Limiting Post-Election Audit.* Recupedado de MIT Computer Science & Artificial Intelligence Lab: https://people.csail.mit.edu/rivest/pubs/Riv17b.pdf

Schneier, B. & Kelsey, J. (1998). Cryptographic Support for Secure Logs on Untrusted Machines. *7th Conference on USENIX Security Simposium.* San Antonio, TX: USA.

Sandler, D., Derr, K., Crosby, S. & Wallach, D. (2008). Finding the Evidence in Tamper-evident Logs. *3rd International Workshop on Systematic Approaches to Digital Forensics Engineering.*

Stanislevic, H. (2006). *Random auditing of e-voting systems: How much is enough?* National Coalition for Election Integrity, E-Voter Education Project.

Stark, P. B. (2008). Conservative Statistical Post-Election Audits. *The Annals of Applied Statistics, 2* (2), 550-581.

Stark, P. B. (March 2010). *Why small audit batches are more efficient: two heuristic explanations.* Recuperado de University of California, Berkeley - Department of Statistics: https://www.stat.berkeley.edu/~stark/Preprints/smallBatchHeuristics10.htm

Stark, P. B. (2015). *Tools for comparison risk-limiting election audits.* Recuperado de University of California: https://www.stat.berkeley.edu/~stark/Vote/auditTools.htm

Verified Voting (2017). *State Audit Laws Searchable Database*. Recuperado de State Audit Laws Searchable Database: https://www.verifiedvoting.org/state-audit-laws/

VVSG (December 2005). *Election Assistance Commision.* Recuperado de Election Assistance Commision: https://www.eac.gov/assets/1/28/VVSG.1.0_Volume_1.PDF

Yasinsac, A. & Bishop, M. (Mayo de 2008). The Dynamics of Counting and Recounting Votes. *IEEE Security and Privacy, 6* (3), 22-29.

[i] These are classified as direct verification systems, separate process systems, token systems, or end-to-end encryption verification systems.
[ii] Individual verification refers to the possibility that each voter has to verify that their vote was recorded correctly.

* Víctor Morales Rocha has a doctorate degree and works at the Institute of Engineering and Technology of the Autonomous University of Ciudad Juárez (UACJ). Coordinator of the National Laboratory of Information Technologies, UACJ campus.
** Oscar Ruiz Hernández is a student of the Master's Degree in Applied Computing from the Autonomous University of Ciudad Juárez.
*** Luis Felipe Fernández Martínez has a master's degree and works at the Institute of Engineering and Technology of the Autonomous University of Ciudad Juárez. Coordinator of the Unit of Knowledge Engineering and Software Engineering, of the UACJ.