SecureBallot: A Secure Open Source e-Voting System

Article

Accepted version

Agate V., De Paola A., Ferraro P., Lo Re G., Morana M.

In Journal of Network and Computer Applications

Publisher: ELSEVIER

SecureBallot: A Secure Open Source e-Voting System

Vincenzo Agate[a], Alessandra De Paola[a,b], Pierluca Ferraro[a], Giuseppe Lo Re[a,b], Marco Morana[a,b]

[a] *Department of Engineering, University of Palermo*
[b] *Cybersecurity National Lab CINI - Consorzio Interuniversitario Nazionale per l'Informatica*

Abstract

Voting is one of the most important acts through which a community can make a collective decision. In recent years, many works have focused on improving traditional voting mechanisms and, as a result, a wide range of electronic voting (e-Voting) systems have been proposed. Even though some approaches have achieved a proper level of usability, the main challenges of e-Voting are essentially still open: protect the privacy of participants, guarantee secrecy, anonymity, integrity, uniqueness, and authenticity of votes, while making e-Voting as trustful as voting. In order to address this issue, we present SecureBallot, a secure open-source e-Voting system that completely decouples the voter identification and voting phases by means of proven cryptographic technologies. The effectiveness of SecureBallot is demonstrated both theoretically, by presenting a formal verification of the whole protocol and assessing the security properties of its software components, and practically, by proposing a case study of university elections that contains all the challenges of a generic voting process.

Keywords:
e-Voting, Privacy, Data Security

1. Introduction

In democratic countries, elections are the fundamental mechanism for citizens to express their opinions and choose their representatives. Since ancient times, voting systems have been the basis of decision-making within communities, inevitably reflecting all societal changes. Indeed, starting with the counting of raised hands or pebbles, in ancient Greece, voting systems evolved by introducing voting booths, paper ballots, punch cards and optical scanning machines.

All these changes were aimed at improving the efficiency and security of voting operations. In the modern days, cyber security techniques can provide a valuable contribution by making the whole process fully transparent and verifiable, speeding up the voting and counting operations, ensuring the privacy of users, as well as guaranteeing the integrity of votes.

Although the terms *e-Election* and *e-Voting* are sometimes used interchangeably, they have distinct meanings, as defined by Meier (2012). The former refers to the overall electronic procedure for electing representatives of a community; conversely, *e-Voting* denotes a generic electronic procedure used by people to express their opinion on various topics, whether major or minor.

In addition to increased reliability and cost savings, e-Voting systems can be easily designed to provide support to users (such as voice assistance for visually impaired people) and to achieve better scalability for large elections.

However, despite their undoubted advantages, the use of e-Voting in real elections is still under debate (Avgerou et al., 2019; Tonkiss, 2009; Alvarez et al., 2011; Falleti and Lynch, 2009). The adoption of an e-Voting system, in fact, involves several challenges. Among these, one of the most relevant is to authenticate users, so that only those entitled to vote can do so, while simultaneously preserving their anonymity, thus ensuring complete privacy and secrecy of votes. To satisfy both of these conflicting goals, any connection between voters and their vote must be prevented.

Apart from technological challenges, gaining the trust of voters seems to be one of the most difficult problems to solve. Considering the decisive influence of elections on the affairs of a community, it is understandable why people are reluctant to changes, and how they are wary of anything they cannot directly monitor and verify. After all, if traditional voting systems have been in place for centuries, why should we change them now?

For this reason, considering that users' positive perception of a system is essential, our aim is to create an e-Voting system that maintains desirable characteristics of traditional elections, improving their security and usability. For example, we decided to retain the necessity of voting at a physical polling station, albeit digitally. This allows us to employ certified staff members that act as

Email addresses: vincenzo.agate@unipa.it (Vincenzo Agate), alessandra.depaola@unipa.it (Alessandra De Paola), pierluca.ferraro@unipa.it (Pierluca Ferraro), giuseppe.lore@unipa.it (Giuseppe Lo Re), marco.morana@unipa.it (Marco Morana)

election officials who securely authenticate users, as well as ensuring the proper running of all voting operations. On the other hand, the use of an electronic system allows voters to freely choose the polling station where to cast their preferences. Such an operation would be very difficult in a traditional system, due to the lack of a central database that keeps track of who has already voted.

To this end, we propose SecureBallot, a secure open source e-Voting system that leverages state-of-the-art cryptographic techniques to protect the privacy of participants and guarantee secrecy, anonymity, integrity, uniqueness and authenticity of votes. SecureBallot exploits a new security protocol that ensures all the above requirements and, at the same time, lets us adopt technologies that are familiar to users. This, in turn, increases their confidence in the system and in the concept of e-Voting on the whole, as we will show in the remainder of the paper. We also test the validity of our system both theoretically, analyzing its security properties, and practically, through an extensive case study of real elections.

A distinctive characteristic of SecureBallot is being open source. We argue that one of the key reasons for distrusting current e-Voting systems is that many existing solutions are based on proprietary software, which cannot be analyzed nor verified. On the contrary, open source software can be accessed by anyone interested, whether they are cyber security experts or simple users. We think this is especially important for e-Voting systems, given their unique characteristics, as discussed in (Moynihan, 2004). For this reason, we decided to release the source code of our e-Voting system, as well as the encryption scheme, making them publicly available. The code is freely accessible on GitHub, on our repository[1]. As a consequence, SecureBallot gets three main benefits: better security, transparency, and the ability to tailor the system to specific demands. It is well known that *security through obscurity* is not a good policy to adopt. If the security of an e-Voting scheme depends on the secrecy of its protocol, which is then disclosed to the public, the damage is irreparable. On the contrary, if a cryptographic scheme is secure despite being publicly known, it is much more reliable. In addition, transparency means that open source code can be analyzed by security experts from all over the world, who can find bugs much faster, inevitably leading to a more secure system. Even those who are unable to verify the code personally, feel more reassured that the community can continuously monitor and improve the development of the system.

Finally, as a result of being open source, SecureBallot is easily extensible and adaptable to the particular needs of different communities. Considering that voting procedures may vary even drastically, in different contexts, the ability to easily modify the voting system is very important and makes SecureBallot reusable in a wide variety of scenarios.

As stated above, one of the biggest challenges in designing e-Voting systems is preserving voters' privacy. Our system strictly separates the identification phase from the voting itself, so that no information about voters is stored or collected in any way. To this aim, we adopt a token-based approach (Sampigethaya and Poovendran, 2006; Agate et al., 2020), introducing the idea of virtual users: polling stations randomly assign temporary identities to users, allowing them to vote in one of the available voting booths, which is also randomly chosen. Virtual users' tokens are constantly reused for different voters, thus ensuring their privacy.

In order to extensively test our system we have chosen university elections as a case study. This allowed us to evaluate the system with different types of elections, involving a variable number of participants, under increasingly challenging conditions. Indeed, elections in a university context often involve heterogeneous groups of participants, from students to faculty and staff members. We think that universities are an excellent testing ground for an e-Voting system, since the digitization of voting operations would bring many benefits and, at the same time, the desire to experiment new ways of voting is greater in universities than in other contexts. This is especially true today, given the Covid-19 pandemic that has led many universities to perform lectures, exams and even graduation cerimonies remotely, exploiting video conference platforms. Given the novel and increasingly urgent necessities that society has to address in the present times, the use of an e-Voting system in a university context may represent an important step forward, and we remark the relevance of experiments carried out in this scenario.

All this motivates our work and results in the following novel contributions.

- We propose SecureBallot, a secure e-Voting system that completely separates the voter's identification and voting phases, exploiting well-known and well-tested security technologies that are familiar to users.
- We introduce a novel secure protocol that guarantees secrecy, anonymity, integrity, uniqueness and authenticity of votes, while using state-of-the-art cryptographic techniques to protect the privacy of users and ensure the secrecy of votes.
- We formally demonstrate the security of the proposed protocol by using an automatic tool (namely Casper/FDR), to verify several security properties such as secrecy and privacy of voting packets and the mutual authentication between parties involved in the protocol.
- We have tested SecureBallot thoroughly in the scenario of university elections, in increasingly challenging conditions. Our system has been extensively used during a period of six months at the University of Palermo, both on mock and real elections.
- We have collected and analyzed feedback from hundreds of heterogeneous users across multiple univer-

[1] https://github.com/ndslab-unipa/SecureBallot

sity elections, by administering special questionnaires to voters.

- We have released the source code of SecureBallot, which is publicly available and can be tested by security experts.

The remainder of the paper is organized as follows. Related work is outlined in Section 2. The system architecture is described in Section 3, while the security properties of SecureBallot are discussed in Section 4. Section 5 gives details of our e-Voting system, analyzing the operations performed before, during and after the voting procedure. A formal verification of SecureBallot is proposed in Section 6. We present the case study of university elections in Section 7. In Section 8, we discuss the scalability of SecureBallot in the case of large elections. Finally, we draw the conclusions in Section 9.

2. Related work

The idea of *e-Voting* systems based on ICT and cryptographic techniques has attracted significant interest from both the academic community and public authorities (Carter and Bélanger, 2005; Moynihan, 2004).

The challenge of providing a suitable e-Voting solution has been addressed by many researchers from different perspectives. Two main e-Voting categories can be identified:

- unsupervised voting, also known as remote electronic voting, which allows users to vote remotely, through the Internet;
- supervised voting, which requires physical polling stations monitored by election officials.

Although both categories have been widely explored and many systems have been tested in real cases with various degrees of success, existing solutions present some limitations, making the choice of the most suitable system for a specific scenario very challenging.

Remote e-Voting allows users to vote without being physically present in supervised environments. This possibility is generally more attractive to users because voting from home is very convenient. Unfortunately, there are two main problems with this approach.

The first concerns the need for voters to trust the remote voting software application and to be confident that their vote is actually counted. Such issue applies to both remote and supervised e-Voting. Currently, this is sometimes solved by using a mechanism that allows users to verify that their vote has been counted correctly; for example, remote e-Voting systems often make use of virtual receipts. The same problem is solved in some supervised voting systems, such as direct-recording electronic (DRE) voting machines (Bederson et al., 2003; Kohno et al., 2004) by means of Voter-Verified Paper Audit Trail (VVPAT) (Villafiorita et al., 2009; Hall, 2006; Carroll and Grosu, 2009). While VVPATs help users to have more confidence that their vote has been counted correctly, they raise potential privacy concerns: if there is an association of any sort between the vote and the voter, users' privacy may be severely compromised. For this reason we do not use VVPATs in any form, avoiding any connection between voters and votes. Nobody (not even the users themselves) is able to prove anything about the votes cast.

The second problem concerns the *incoercibility* of electors. All voting systems, whether electronic or paper-based, must ensure that users can vote freely, without any external influence. Otherwise, users might be forced to vote in a certain way, or deliberately decide to sell their vote to the highest bidder. To guarantee incoercibility, it is essential that users are not spied on while they are voting, and that it is impossible for them to provide proof of their vote to third parties. In fact, if it were possible for users to prove who they voted for, an external attacker could force them to show such proof.

Remote e-Voting systems cannot fully guarantee the incoercibility of voters. Indeed, for a remote system, it is impossible to ensure that users are alone at the time of voting: a third party could witness the vote, unbeknownst to the e-Voting system. Thus, by definition, it is impossible to guarantee incoercibility in remote e-Voting systems, as demostrated by Wang et al. (2017).

On the other hand, remote e-Voting generally guarantees *receipt-freeness*, i.e. users cannot prove their vote after having cast it. However, remote e-Voting systems cannot guarantee users' privacy *during* the actual voting, as argued above. The receipt-freeness property is thus weaker than incoercibility; the latter implies receipt-freeness, but the reverse is not true, as shown in (Sampigethaya and Poovendran, 2006).

Despite these limitations, and given their great convenience, many researchers have focused their efforts on developing new Internet-based electronic voting schemes that allow voters to participate remotely (Wu et al., 2014; Rezvani and Hamidi, 2010; Agarwal and Pandey, 2014; Tornos et al., 2013; Yi and Okamoto, 2013), using ad-hoc web applications.

Civitas (Clarkson et al., 2008) is one of the first electronic voting systems that allows users to vote securely from a remote client, while providing anonymity, integrity and receipt-freeness. To guarantee receipt-freeness, voters may use their private key and run an algorithm to generate fake credentials. Voters provide these credentials in case an attacker tries to influence their votes. However, the attacker might force users to reveal their credentials, bypassing this anti-coercion measure.

Helios (Adida, 2008) is one of the most famous Internet-based remote voting systems that uses a web browser. Users' votes are submitted only after being encrypted, so the secrecy of votes is ensured. Helios makes no attempt to solve the problem of coercion, so it is only usable for elections with a low coercion risk. In other works, such as (Ahmad et al., 2009), web applications are replaced by

apps running on mobile devices.

The same general structure provided by Helios was recently adopted by Ordinos (Küsters et al., 2020), which significantly extends the capabilities of its predecessor by introducing the concept of tally-hiding. The approach proposed by the authors makes it possible to get the results of an election without revealing the full tally (e.g., revealing only the winner, a ranking between candidates or the k best/worst candidates). While these features are indeed interesting for specific contexts, such as in the case of very small elections where the results may embarrass the participants, Ordinos does not fit well in scenarios where transparency about the election results are a prerequisite, e.g., political elections. Furthermore, Ordinos still suffers from the issues common to any unsupervised voting system.

Incoercibility is hard to guarantee even for supervised e-Voting systems. Sampigethaya and Poovendran (2006) further classify e-Voting systems in three categories. In *hidden vote* systems, users openly submit encrypted votes; in *hidden voter* systems, users anonymously submit plaintext votes, while in *hidden voter with hidden vote* systems, users anonymously submit encrypted votes. The study shows that incoercibility can only be guaranteed if there is no link between the vote and the voter, and this can only occur in supervised *hidden voter with hidden vote* systems, as SecureBallot. Indeed, our e-Voting system guarantees the incoercibility of voters, as will be demonstrated in Section 4, making it a more suitable choice than remote e-Voting systems for all elections where user privacy is paramount.

In the context of supervised e-Voting, two different types of systems are frequently used: DRE voting machines and computerized voting systems (Ochoa and Peláez, 2017).

DRE systems depend on proprietary hardware devices, which are often costly. Plus, it is not uncommon for such devices to exhibit vulnerabilities that jeopardize secrecy, privacy and the proper running of elections (Frankland et al., 2011). The alternative is to use common PCs or laptops as voting stations, running specially designed software. The advantage of using custom-made systems is that they can be tailored to specific needs, and meet any type of requirement.

The adoption of such solutions is very convenient in medium or large scale scenarios, where the cost of acquiring new hardware can be amortized over multiple elections. In a university setting, for instance, this sounds like the most convenient solution, and the software may be developed taking into account the peculiar characteristics of such a scenario.

Bingo Voting (Bohli et al., 2007) is a supervised voting system that provides users with a receipt that associates a random number to the chosen candidate. Other candidates on the receipt are assigned different random values extracted from a pre-calculated codes pool. The user can then verify that their code is included in the count during the tallying phase. Differently from other systems that use a receipt, this approach does not provide other more advanced services and has the disadvantage of not fully removing the link between the vote and the voter.

Other studies (Volkamer and McGaley, 2007; Neumann, 1993; Gritzalis, 2002) have highlighted standard criteria of security, privacy, freedom and universality that all e-Voting systems should guarantee. The formal definition of these criteria, as well as their enforcement, are essential to design and develop voting systems that can be used in a real context.

Recently, major research efforts have focused on the use of blockchains for the implementation of voting systems (Hanifatunnisa and Rahardjo, 2017; Kshetri and Voas, 2018; Hjálmarsson et al., 2018; Mustafa and Waheed, 2020; Aggarwal et al., 2019).

The key idea behind using blockchain technology as the basis of a voting system is to endow voters with a *wallet*. By analogy with "virtual currency", each voter receives a single *coin*, representing a voting opportunity, that can be transferred to the candidate's wallet to express a vote (Kshetri and Voas, 2018). However, the use of this technology often raises concerns, since it is still unfamiliar to users. Moreover, besides the trust issues, blockchains require a lot of energy to perform essential operations such as authentication and block validation, and are quite slow. Therefore, using such a technology for large-scale elections may not be practical yet (Kshetri and Voas, 2018).

Voatz[2] is a proprietary voting system based on mobile phones that leverages a combination of permissioned blockchain, biometrics, and hardware-supported keystores to provide end-to-end encrypted, voter-verifiable voting. Despite Voatz has been used to conduct several real elections in the US, Specter et al. (2020) highlighted that details about its implementation were never provided and discovered several threats that led a Washington county to abandon its usage.

Zissis and Lekkas (2011) investigated the advantages and disadvantages of adopting a cloud solution for e-government information systems. A mixture of cloud computing and encryption techniques can be used to tackle a number of typical e-Voting system threats: integrity, confidentiality, authenticity, and availability of data and communications. Unfortunately, the use of this technology seems to be far from real use as people often do not trust the management of data in the cloud (Xiao and Xiao, 2012).

The family of Prêt-à-Voter voting systems (Ryan et al., 2009) focuses on end-to-end verifiability (Ryan et al., 2015), i.e., voters can verify at any time if their vote has been counted correctly, using a receipt. Prêt-à-Voter is similar to traditional voting systems, and it even uses paper ballot cards. As such, it does not benefit from some of the classic advantages of e-Voting. Plus, it presents multiple vulnerabilities, such as voting chains, as shown by Ryan and Peacock (2010). In these attacks, someone smuggles

[2] https://voatz.com/

Table 1: Comparison of the main voting systems proposed in the literature.

	Supervised	Hidden voters with hidden votes	Receipt-freeness	Incoercibility	Open-source
Bingo Voting	✓	✓	✓	✓	-
Helios	-	-	✓	-	✓
Prêt-à-Voter	✓	✓	✓	-	-
Civitas	-	✓	✓	-	-
Ordinos	-	-	✓	-	-
Voatz	-	✓	✓	-	-
SecureBallot	✓	✓	✓	✓	✓

an unused ballot card out of the polling station and marks his chosen candidate. Then he passes the filled-in ballot paper to the next voter he wants to influence. The user submits the already filled-in ballot paper, returning the empty one.

Moreover, such systems need to trust election authorities unconditionally with secrets that might undermine users' privacy. However, even if election officials are trustworthy, it is always possible that someone may leak information during the distribution or storage of ballots, thus exposing the connection between voters and ballot papers, and undermining the secrecy of votes.

Other works presented e-Voting systems that offer a user experience similar to that of traditional ones, improving user satisfaction by increasing their perception of integrity, security, and trust (Yao and Murphy, 2007; Bélanger and Carter, 2008; Antoniou et al., 2007; Avgerou et al., 2019; Khamitov et al., 2019). Pomares et al. (2014) carried out a study of characteristics that may improve user confidence, such as the usability of the system. When an e-Voting system is too different from traditional ones, users may be wary of the underlying mechanism.

Further information about all the e-Voting systems we discussed are available in multiple surveys (Wang et al., 2017; Sampigethaya and Poovendran, 2006; Yumeng et al., 2012).

Table 1 summarizes the most relevant characteristics of several voting systems belonging to the categories previously described, from supervised ones, such as Prêt-à-Voter, through those that allow users to vote remotely, ranging from Helios and recent systems like Ordinos, to blockchain-based applications such as Voatz.

By comparing SecureBallot with other systems proposed in the literature, it results that one of the advantages of our solution is guaranteeing the property of incoercibility. Table 1 highlights that most other systems underestimate this important property, meeting only its weaker version (receipt-freeness). However, we think that this property is crucial for e-Voting systems where privacy requirements are strict.

The same table also shows that SecureBallot is one of the few e-Voting systems that is open source, as Helios. This point is critical, since one of the reasons why users distrust e-Voting systems is that applications based on proprietary software cannot be easily analyzed and verified by experts and common users alike. A prime example of this problem is represented by Voatz, whose major security flaws have been identified through reverse engineering, given the complete lack of source code and proper documentation available (Specter et al., 2020). In open source systems such as SecureBallot, continuous analysis of the source code by security experts would allow detection of potential problems much earlier, resulting in increased user confidence.

From a feasibility standpoint, some unsupervised systems exhibit advantages over supervised ones in terms of a greater convenience in voting remotely, time saving, as well as reduced hardware purchases. However, as argued previously, remote e-Voting systems cannot guarantee the same strict security properties necessary for the proper conduct of nationwide elections. Ronald L. Rivest himself addressed this topic recently, harshly criticizing Internet voting and blockchain-based systems (Park et al., 2021).

Unlike previous work, SecureBallot focuses on gaining users' trust, since a system that lacks public confidence will never really be used on a large scale. To do this, we exploit technologies familiar to users, while simultaneously providing critical security features necessary for elections. This is confirmed by our own real-world testing, which will be presented in Section 7. Taking advantage of the continuous feedback we received from users, we have improved SecureBallot more and more, until we have reached a very high degree of satisfaction. On the other hand, many other e-Voting systems in the literature remain purely theoretical and have never been tested in the field.

3. Architectural overview

In this section we present the hardware and software components of SecureBallot, and the roles of the users and staff members who are involved (i.e., the actors of the system). SecureBallot belongs to the category of voting computers or electronic voting machines, which operate through distributed *polling stations* connected to a central *virtual ballot box*, that collects the preferences expressed by voters in encrypted form.

This class of digital voting systems involves the use of PCs, laptops, tablets and other off-the-shelf devices. The proposed solution is also different from remote e-Voting and web-based applications, since it requires the presence of staff members (e.g., representatives of electoral authorities) on site, in order to verify both that users have the right to vote and that voting operations are carried out correctly. Such a supervised approach makes it possible to meet more strict security requirements as compared to remote web-based systems, as we will discuss in the next sections.

3.1. Actors of the system

In order to present the features of SecureBallot, two different types of actors who interact with the system are considered: voters and staff members. Any person who has the right to vote is a valid *voter*. All voters are previously inserted in a centralized list, accessible by staff members at any polling station; thus, it is always possible to check whether a person actually has the right to vote and whether he has already voted or not.

Staff members can be categorized into three groups, depending on their role and the election phase they are in charge of:

- staff members involved in preliminary operations, which involve preparing and checking the digital list of voters, creating the digital voter cards, as well as generating all secret keys that will be necessary to activate the voting stations on election day;
- staff members in charge of physically supervising all voting operations; their tasks include verifying the proper functioning of the voting systems, setting up the voting stations using the previously mentioned secret keys, verifying the identity of users and registering them before they are allowed to vote;
- *election supervisor*, who oversees the tallying operations, digitally signs the results and publishes them.

SecureBallot requires that all voting operations are encrypted with asymmetric schemes and that the necessary keys for each election are generated by a public official (e.g. a notary public or police officer) who ensures the proper conduct of voting operations. The public key of the current election is used as a *KEK* (key encryption key), while the corresponding private key will be kept secret by the public official until the tallying phase, so that no one can interfere with the proper conduct of the election.

3.2. Hardware and software components

SecureBallot can use readily available PCs, laptops and tablets so as to reduce organizational costs. Each of these devices is equipped with software developed specifically for the digital voting system.

The overall system includes several software applications that run simultaneously on multiple physical machines, as shown in Figure 1. In particular, the proposed architecture involves the use of:

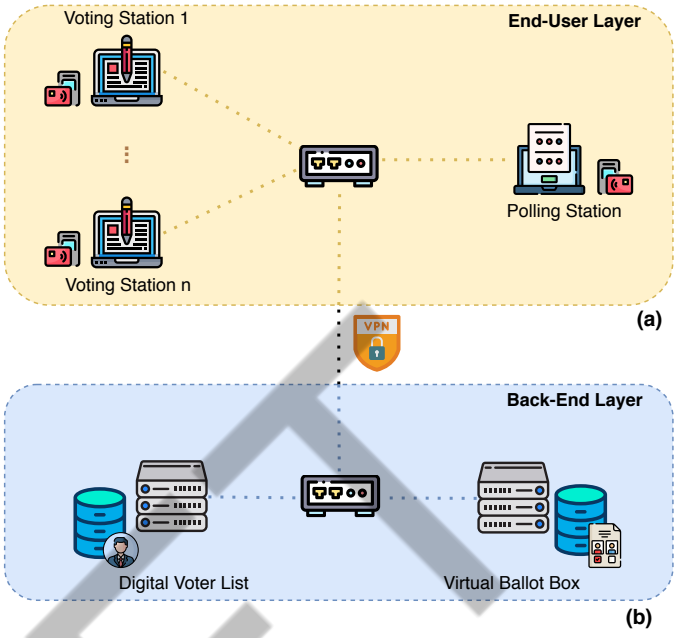


Figure 1: Hardware architecture overview.

- a software application, called *polling station*, used by staff members to monitor voting operations and verify whether users are entitled to vote;
- a software application installed in each voting booth, called *voting station*, that allows users to espress their preferences by inserting their digital voting card in a virtual ballot box;
- a centralized software application that acts as a *virtual ballot box*, which collects votes cast by users in encrypted form.

Voting involves continuous interaction between these software applications; in order to guarantee a suitable security level at the network layer we decided to connect all previously described system components via a dedicated VPN (Virtual Private Network).

Figure 1-a shows the polling stations and the voting stations, which, being directly used by voters and staff members, require to be user-friendly, resilient and fail-safe. After the initial identification, each voter receives a temporary *unlocking token*, which allows him to *unlock* his designated voting station and vote. In SecureBallot, we used NFC tags as unlocking tokens, but the system is compatible with similar products that use symmetric encryption to guarantee authentication and data integrity. Such tokens are completely anonymous and are associated only with *virtual* users, as will be explained in Section 5. SecureBallot does not link tokens to real users in any way, in order to guarantee their privacy.

Figure 1-b shows the back-end of the system, consisting of the *virtual ballot box* that contains all encrypted votes

and the database with the list of people authorized to vote, named *digital voter list*.

The virtual ballot box is the most sensitive component of the system, as it is responsible for receiving all encrypted votes sent by the polling stations and storing them securely and reliably till the tallying phase. For this reason, the virtual ballot box adopts state-of-the-art cryptographic tools to ensure the authenticity and confidentiality of voting packages, such as symmetric and asymmetric encryption, as will be discussed extensively in Section 5. It is worth noticing that users can vote only if the connection between voting station and ballot box is up and running, because sending a vote and storing it in the ballot box are considered as a single atomic operation, which is handled by using distributed database transactions.

In order to ensure the complete anonymity of all voters, the virtual ballot box does not receive any information about users. The only data handled by the virtual ballot box are those relating to anonymous voting cards, that are encrypted with a symmetric key randomly generated at runtime, which in turn is encrypted with the election's public key that serves as *KEK*.

The *digital voter list* that contains the list of voters is intended to maintain consistency between different polling stations.

4. Security properties

Given the importance of elections, all e-Voting systems are expected to meet specific security properties, as discussed in (Gritzalis, 2002; Volkamer and McGaley, 2007). In this section, we show that SecureBallot meets all of these properties, demonstrating how each one is satisfied.

Secrecy: one of the most important constraints of any voting system is *secrecy*. Votes have to be cast by ensuring that no one is able to associate the preferences expressed with the corresponding voter.

In an e-Voting system, this means that even if adversaries were able to intercept the payload exchanged between different system components, it should be impossible to infer the preferences expressed by specific voters.

For these reasons, we use a two step encryption process which entails both symmetric and public key cryptography, namely Advanced Encryption Standard (AES) in CBC mode and RSA. At first, each vote is encrypted with a symmetric key, which is randomly generated for each voting packet. Then, the random key is encrypted with RSA by using the public key of the current election. At the beginning of the tallying phase, the election's private key is used to decrypt each symmetric key and, as a result, the content of the corresponding voting cards.

Using this encryption mechanism, the only person who can start the tallying phase is the election supervisor, assisted by a notary who, after the end of the election, officially discloses the necessary private key.

SecureBallot satisfies the secrecy property, as demonstrated by the formal analysis presented in Section 6. Notably, the full analysis will show that the secrecy of voting is maintained in all cases, and cannot be violated even if the attacker is aware of secret keys that should be known only to the voting station and the central ballot box.

Privacy: users' *privacy* is guaranteed if it is not possible in any way to trace a vote back to the voter who cast it. To ensure that, our system completely decouples the identification and voting phases and does not collect any information about users. During the entire voting process, no system component knows both the identity of voters and their votes. In the formal analysis we will show that SecureBallot satisfies this property and that each vote is completely unrelated to both the voter and the voting station it was sent from.

At the identification phase, staff members use the polling station software to identify users and verify that they have not already voted. Thus, the polling station knows the identity of voters, but not the preference they will express. Conversely, the voting station, which allows the user to vote and encrypts the resulting voting package, will have no information about the identity of the voter.

The central ballot box receives encrypted and anonymous voting packages, hence it knows neither the identity of the voters nor the preferences expressed. Finally, during the tallying phase, the content of each voting packet is decrypted, but it is impossible to trace back to the original voter, since there is no link between the vote and the voter.

Eligibility to vote: unlike traditional paper-based systems, SecureBallot allows users to vote indifferently in any polling station while also guaranteeing each entitled person to vote just once. To this aim, a digital list of voters is checked by staff members, that are also responsible for providing valid voters with unlocking tokens (such as NFC tags), which are required to enable the voting station.

The digital voter list is immediately updated when each vote is completed and successfully stored in the virtual ballot box, exploiting distributed database transactions. By checking this list, all polling stations are able to verify if users are valid voters, while also preventing them from voting multiple times (i.e., *double voting*).

Authenticity: a vote is authentic if it comes from a reliable source or, in other words, if it is submitted by one of the authorized voting stations. In order to establish whether a vote has actually been cast in one of the authorized booths, our system authenticates the voting package. This ensure that attackers cannot falsify votes. A genuine voting package is authenticated by the voting station through a Hash-based Message Authentication Code (HMAC) with SHA-256 digest. The secret key required by HMAC is derived from a passcode entered by staff members when activating each voting station on election day. Once the virtual ballot box receives the vote package, it

verifies the HMAC code attached, discarding forged votes and storing genuine ones. Moreover, in order to guarantee the authenticity of votes until the tallying phase, the virtual ballot box digitally signs all vote packages before storing them. The formal analysis presented later on will prove that SecureBallot fully meets this property. Indeed, the analysis shows that the use of HMAC and digital signature is critical. If they were not used, it would not be possible to ensure that every vote came from an authentic voting station and that all encrypted votes saved in the database are unaltered.

Integrity: the integrity of votes is guaranteed at different levels of the architecture. At the communication level (above TCP), the integrity of all the data transmitted from the voting station to the virtual ballot box is guaranteed by TLS. This protects the communicating parties from any attempt to tamper, intercept and falsify the transmitted packages. The virtual ballot box digitally signs all received voting packages before storing them in the database, thus ensuring their integrity. SecureBallot guarantees this property, as will be demonstrated in the formal protocol analysis. In particular, the use of a secure communication channel makes the protocol resilient to attacks even if some of the secret keys are compromised.

Incoercibility: this property requires that users cannot be influenced in any way while voting. External influences include either being compelled to vote for a specific candidate, or selling one's vote for money.

Note that, as detailed in the related work section, most e-Voting systems do not meet this property. On the contrary, SecureBallot guarantees incoercibility by adopting the same controls as traditional elections, i.e., staff members supervising the procedure and voting booths that guarantee users' privacy.

Moreover, incoercibility requires that users cannot prove to others that they have voted in a certain way (*receipt-freeness*). This is guaranteed in SecureBallot by the absence of receipts. Note that the property of incoercibility implies receipt-freeness, but the reverse is not true, as argued by Sampigethaya and Poovendran (2006).

Validity: a major challenge in conventional paper-based voting systems is handling invalid votes, which can be caused by voter negligence.

Since SecureBallot guides users during the submission phase, the voting form is always filled in a valid and unequivocal manner. The user can choose to express the maximum number of preferences allowed by the voting card, or leave it deliberately empty. In all cases, SecureBallot asks for confirmation before actually sending the vote. Finally, the system also ensures that the vote submission has been completed successfully, by exploiting distributed database transactions.

Transparency: maintaining *transparency* during voting operations means that voter and staff members must have a confirmation that the vote expressed has been successfully and securely stored in the virtual ballot box. To this aim, the system communicates the completion of the operation to both the voter and the staff members. This is achieved by means of a message displayed on the voting and the polling stations. Otherwise, the system alerts both the voter and the staff, allowing them to repeat the voting operations.

Accuracy: this property is guaranteed if all the voting packets are properly counted and never lost. SecureBallot uses distributed database transactions to group the most critical operations when submitting votes to the virtual ballot box. As a result, if any errors occur while voting (e.g. connection lost), all pending operations are aborted and the user is allowed to vote again, as specified by the *transparency* and *validity* properties.

Verifiability: the verifiability property can be examined from two different points of view: individual verifiability and global verifiability. *Individual verifiability* means that each voter can be confident that his vote has been correctly stored and, subsequently, counted during the tallying phase. This is guaranteed in SecureBallot by the fulfilment of two other properties described above, namely *transparency* and *accuracy*.

Global verifiability ensures that all voters, as a whole, are confident in the proper conduct of electoral operations, and in particular that all votes cast have been stored and subsequently counted, and that only those who were entitled to vote actually did so. This is guaranteed in SecureBallot by the fulfilment of three other properties, namely *eligibility to vote*, *transparency* and *accuracy*.

5. SecureBallot

Ballot procedures can often be very complex, and their features can vary drastically depending on their intrinsic electoral rules (i.e., laws or regulations set by the company or organization in charge of the voting procedure). Differences often involve the amount of candidates and elected people, the type and number of votes expressed by users, and the way in which candidates may be grouped into parties and electoral groups. In this section we present the technical features that allow SecureBallot to handle all these different demands in a transparent way to users.

5.1. Electoral procedure steps

To deal with the individual characteristics of each election, our system operates in three distinct phases, corresponding to the operations performed before, during, and after the voting procedure, i.e., the *setup*, the *voting phase* and the *tallying*.

Setup: any voting system, whether traditional or electronic, requires a *setup*, during which the system is configured to handle the current election. At this stage, organizers choose the number and location of the polling stations,

Table 2: Symbols used in the complete security protocol.

Symbol	Description
BB	Virtual ballot box
CBC	Cipher Block Chaining
ES	Election supervisor
H	hash value
IV_i	i-th initial value for CBC
IV_iC	Encrypted IV_i
$K_{S_{VS_j,BB}}$	Symmetric key shared between j-th voting station and virtual ballot box
K_i	i-th symmeytric key, used to encrypt a voting packet
K_iC	K_i encrypted with K_E^+
K_{BB}^-	Virtual ballot box's private key
K_E^-	Election's private key
K_E^-C	K_E^- encrypted
K_{BB}^+	Virtual ballot box's public key
K_E^+	Election's public key
MAC_i	MAC of the i-th voting package
N_i	i-th nonce
PKD	public key decryption
PKE	public key encryption
$Sign$	digital signature
$SymD$	symmetric decryption
$SymE$	symmetric encryption
V_i	i-th voting packet
V_iC	i-th voting packet, encrypted with K_i
$V\text{-}HMAC$	MAC verification algorithm
$V\text{-}Sign$	digital signature verification
VS	Voting station

and ballot cards are created and adapted to the number and type of candidates.

SecureBallot simplifies these operations through the use of a dedicated GUI, which can only be accessed by election officials. In particular, our system allows staff members to create and configure the voting cards by specifying the list of candidates and the maximum number of preferences that may be cast. SecureBallot also allows staff members to upload the list of voters to the central database and keep it synchronized during the *voting phase*, i.e. updating the status of those who have already voted.

Other tasks carried out during the *setup* concern the configuration of the voting procedure itself, such as setting the starting and ending date of the voting operations, and configuring the voting stations as nodes of a VPN.

Voting phase: during the *voting phase*, voters can express their preferences. Electoral officers that are in charge of the polling station system are directly responsible for the following tasks:

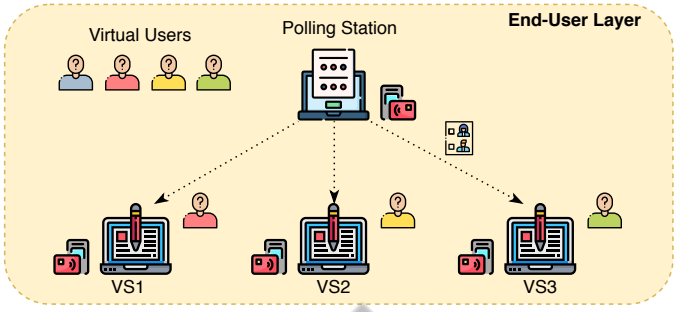


Figure 2: Interactions between polling stations, virtual users and voting stations (VS).

- identifying prospective voters according to the regulations of the current electoral procedure (e.g., through their ID cards);
- verifying whether they are entitled to vote and they have not previously voted;
- using the polling station software to enable one of the voting stations;
- offering voters the necessary explanations and providing them with one of the unlocking tokens;
- ensuring that the voting operation has been completed without any issues, and having the unlocking token returned to them.

Once authorized by an election officer, the voter can enter the assigned voting booth, unlock the voting station through the unlocking token, and choose the preferred candidates. The vote is then encrypted and sent to the central ballot box, anonymously, ensuring the aforementioned security, integrity and consistency requirements. Finally, the voter has to return the unlocking token to the election officers to complete the operation.

Tallying: the final phase, which takes place after the conclusion of the elections, concerns the counting of votes. During this phase, polling stations are shut down, and voters no longer have the opportunity to express their preferences.

At this stage, the election supervisor can start the tallying process, in which the votes are deciphered, counted, and the tallying outcome is saved in XML format and digitally signed by the supervisor. Note that the tallying phase can only be performed in the presence of a notary, who holds the election's private key.

The voting and tallying phases will be described in detail in the following sections, analyzing the data flow, the operations performed by all actors involved and the protocol used to ensure the security properties outlined in Section 4. Table 2 explains the symbols used in the remainder of the paper.

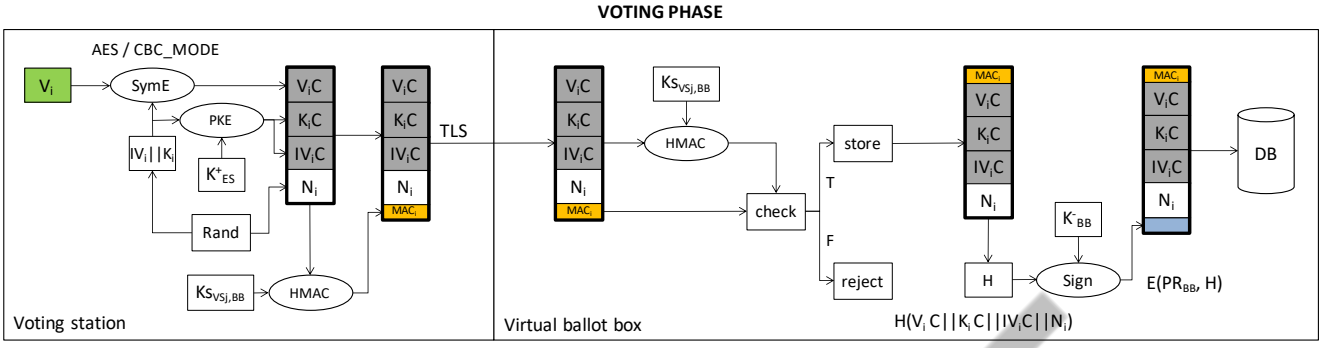


Figure 3: Security schema of the voting phase.

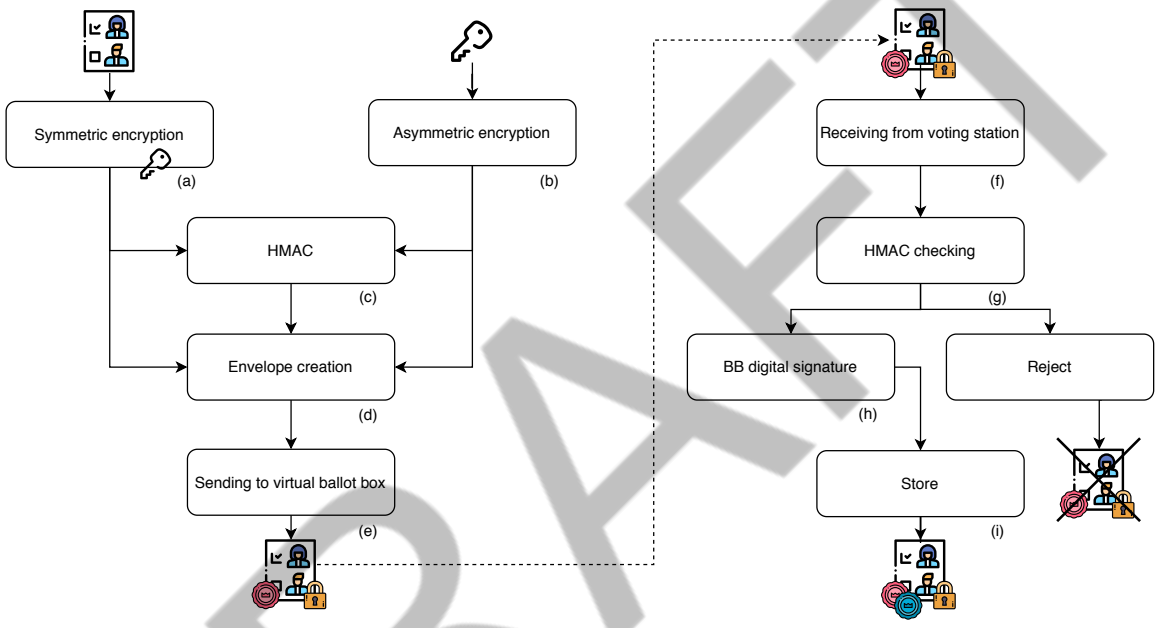


Figure 4: Voting phase operations.

5.2. *Voting phase operations*

Once voter's eligibility has been verified, the system no longer needs information about the user that is voting. We introduce the idea of *virtual users* to guarantee the anonymity of voters, and at the same time ensure that they can vote only if authorized.

In order to allow only authorized users to vote, the system provides that the virtual user v_i, who has been given token t_k, will be able to vote only once in the voting station VS_j. Figure 2 highlights the interactions between virtual users, polling station and voting stations, showing a possible association. Each user can vote only once in the assigned voting station.

Usually there are more virtual users than voting stations, as in Figure 2. In this way, if many users arrive at the polling station at the same time, and the voting booths are all occupied, staff members can still assign them a temporary identity as virtual users. When a voting booth is available, a new virtual user can be associated with that booth. Voting operations can thus happen simultaneously.

However, if the number of virtual users was too high, i.e., more than the number of voters, there would be a unique value (the virtual user id) linked to each user. Although this value would not be a privacy risk by itself, since it is not stored in any way, the goal of SecureBallot is to avoid, by design, any association between vote and voter. Constantly reusing virtual identities from the same pool achieves this goal, ensuring the anonymity of users.

In certain electoral procedures, voters are divided into groups, and depending on the group they are part of, they will be entitled to vote with one or more of the available ballot papers. In other words, the number and type of ballot papers presented to each type of voter may vary. The use of XML ballot papers allows voting stations to support all these options without having to be specifically reconfigured for each election. Virtual users are associated with the type of voter (and therefore with the number and type of ballot papers). The voting station then shows the correct ballots to each voter, albeit not knowing his identity.

To ensure privacy, votes are cast in dedicated voting booths. When users enter the booth to which they have been assigned, they interact with the computer and the token reader located there. Users can enable the voting station software by holding the unlocking token they have received (such as an NFC tag) near the token reader. If the user is in the correct voting booth, his unlocking token will be recognized and the voting station will be activated. Then, voting cards are shown as a form the user can compile and submit. The compiled cards are kept secret and stored in a XML file, that also contains other utility fields (e.g., info about the polling station or the type of election) required to provide administrators with statistics on voter turnout.

Figure 3 shows the security schema of SecureBallot in the voting phase, highlighting the encryption operations and the messages exchanged between voting stations and virtual ballot box, while Figure 4 complements this schema by showing a flow chart of the operations performed at a higher level of abstraction.

Each field of the i-th voting card is encrypted using a symmetric key, K_i, which is automatically generated by the respective voting station. AES symmetric encryption is used in CBC mode to keep the voting card encrypted until the end of the election (Figure 4-a). In our implementation, we used a 128-bit key for AES, but naturally it is also possible to exploit longer keys, such as 192 or 256-bit ones; both block size and initialization vectors (IVs) are 128 bits. SecureBallot generates each K_i and IV_i using a cryptographically secure pseudo-random number generator. In addition, the K_i symmetric key is also ciphered with the election's public key, K_E^+ (Figure 4-b). The resulting key, K_iC, is sent to the central ballot box together with the encrypted voting package and IV_i. In SecureBallot, we used RSA with Optimal Asymmetric Encryption Padding (OAEP) exploiting 1024-bit RSA keys.

Please note, once again, that the private key of the current election is kept secret and inaccessible to everyone until the end of the election. Not even the supervisor can have access to it, and therefore he cannot interfere in any way with the voting operations. Only during the tallying phase an external authority, such as a notary, officially hands over the private key to the election supervisor, who can then start the counting of votes.

Figure 4-c shows how the voting package (encrypted with K_i), the encrypted key K_iC, the encrypted initialization vector IV_iC, and a *nonce*, N_i, are processed by the HMAC algorithm using a SHA-256 digest (HMAC-SHA256), which guarantees their integrity and authenticity:

$$MAC_i = \text{HMAC}(K_{S_{VS_j,BB}}, V_iC \,||\, K_iC \,||\, IV_iC \,||\, N_i) \quad (1)$$

The secret key $K_{S_{VS_j,BB}}$, shared between the j-th station and the central ballot box, is used by the HMAC-SHA256 algorithm to authenticate the message. Such key is derived from access codes used to activate each voting station in the preparatory phase. The nonce N_i is used by SecureBallot to prevent replay attacks. As will be explained below, by modifying this *nonce* and keeping the encrypted V_iC, K_iC and IV_iC unchanged, the voting station can recalculate the HMAC and get a different digest, if required by the central ballot box.

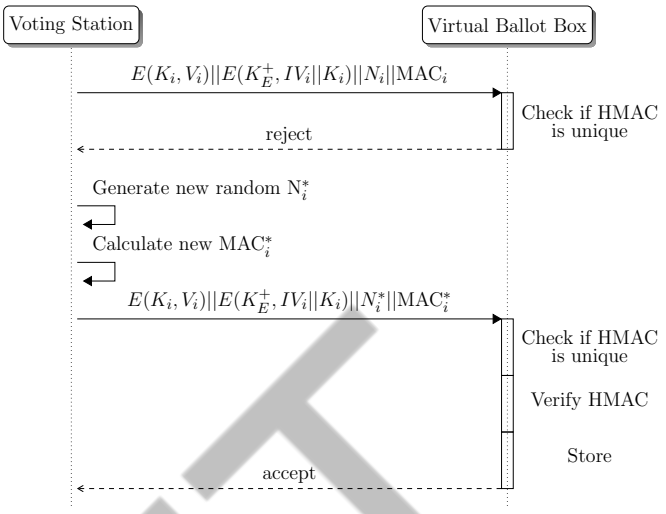


Figure 5: Sequence diagram showing messages exchanged between VS and BB when a duplicate HMAC is received.

The complete voting package, sent from a voting station to the central ballot box, consists of the symmetrically encrypted vote and the asymmetrically encrypted key, as well as the corresponding nonce and HMAC digest (Figure 4-d). The content of the voting package can be expressed as:

$$E(K_i, V_i) \,||\, E(K_E^+, IV_i \,||\, K_i) \,||\, N_i \,||\, MAC_i, \quad (2)$$

where MAC_i is the HMAC digest calculated in Equation 1. Although, as already mentioned, the whole network is protected by IPsec with the creation of the VPN, all communications between polling stations, voting stations and the virtual ballot box are further protected by TLS (Figures 4-e and 4-f), in order to guarantee a suitable security level at the transport layer.

A further security layer is provided by the central ballot box, at the application level. As shown in Figure 4-g, the central ballot box checks the HMAC digest of the received vote packet, as shown in Figure 3:

$$\text{V-HMAC}(\text{HMAC}(K_{S_{VS_j,BB}}, V_iC \,||\, K_iC \,||\, IV_iC \,||\, N_i)) \quad (3)$$

If the verification is successful, it means that the ballot box is unaltered and valid. Indeed, only the central ballot box and the authorized voting stations know the $K_{S_{VS_j,BB}}$ key, which is needed to correctly calculate the HMAC.

In addition, to prevent *replay attacks*, the central ballot box maintains a database of previously received HMAC digests. If the digest of a new voting package is identical to another received in the past, two different scenarios may have happened, namely an attacker is trying to perform a replay attack, or an HMAC collision occurred.

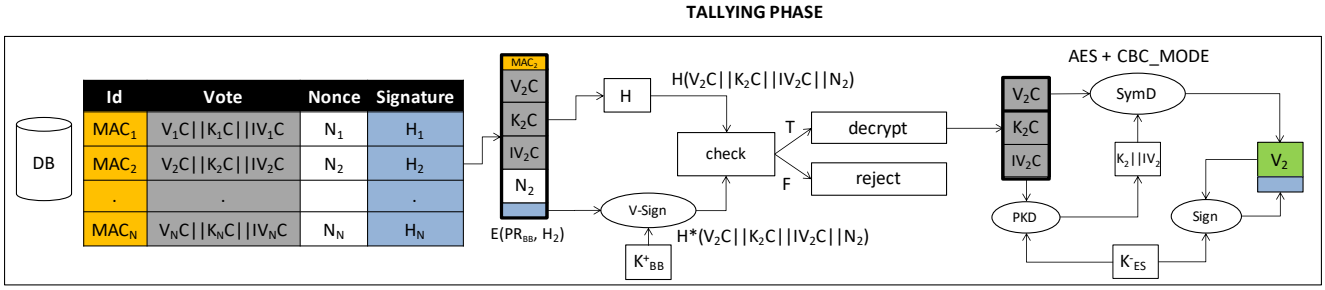


Figure 6: Security schema of the tallying phase.

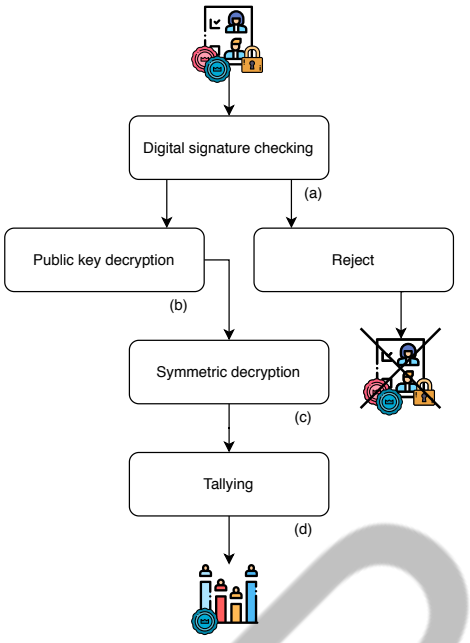


Figure 7: Tallying operations.

In the first scenario, an attacker has somehow intercepted a valid voting package, acting as *man-in-the-middle*, and is trying to re-send it several times, hoping to duplicate a valid vote. This might make sense if the attacker can speculate, with some degree of uncertainty, about the content of a voting package, even without decrypting it. If an attacker knows, from prior knowledge, that votes cast from a certain polling station PS_x have a high probability of being favorable to candidate Y (due to bias in voter distribution), he might be interested in duplicating voting packages from voting stations linked to PS_x, hoping that they are actually in favor of candidate Y. Duplicating many packages like this, the attacker would have a good chance (based on the distribution of voters) to influence an election by favoring the chosen candidate.

The second scenario is an HMAC collision between two different valid voting packages. Although this is a remote eventuality, it is still possible that such a collision might occur; hence, SecureBallot provides an interactive protocol between voting station and virtual ballot box to resolve such conflict. To handle both scenarios, each voting package includes a random nonce, N_i, which is included in the HMAC calculation, as described above.

Figure 5 shows the messages exchanged between a voting station and the virtual ballot box when a duplicate HMAC is received. If the virtual ballot box detects a duplicate digest, the corresponding voting package is rejected, and the voting station is asked to generate a new random nonce, N_i^*, and recalculate the HMAC digest, obtaining a new MAC_i^*. In order to do this, it is not necessary to re-encrypt the vote or the symmetric key: modifying the nonce is sufficient to obtain a new digest, different from the previous one. The reason why SecureBallot uses a random nonce, and not a counter, is to avoid associating a sequence to the votes received, which could affect users' privacy.

If the voting station is legitimate, it will be able to perform these operations and generate a new voting package with the respective digest MAC_i^*, as shown in Figure 5. Conversely, an attacker will not be able to generate a valid HMAC because he does not possess the necessary symmetric key, $K_{S_{\mathrm{VS}_j,\mathrm{BB}}}$.

Indeed, if the HMAC is unique, the virtual ballot box checks the validity of the digest with respect to the voting package. If the check is successful, the central ballot box digitally signs the voting package (Figure 4-h) and stores it in a database (Figure 4-i) until the end of the election. At this point, the virtual ballot box sends a confirmation to the voting station. As described in Section 3, sending, validating and storing a vote is considered as a single atomic operation, handled through distributed database transactions. Finally, no one can modify a vote before the tallying, since it is digitally signed, nor decipher it, as this would require the election's private key.

5.3. Tallying phase operations

After the conclusion of the voting session, the election supervisor starts the tallying process. In a traditional paper-based system, votes are manually counted by the electoral staff, with potential calculation errors, and long waiting times for the results' diffusion. In SecureBallot, the tallying process is very simple, secure and immediate. Figure 6 shows the security schema of the tallying phase, highlighting the cryptographic operations performed by

the virtual ballot box and its interactions with the election supervisor. Figure 7 illustrates the corresponding flow chart explaining the same operations at a higher level of abstraction.

The only person entitled to start the tallying phase is the election supervisor, by requiring access to the private key of the specific election. Indeed, till this moment, the aforementioned private key is maintained secret by a notary or public official, which plays a warranty role. This figure, after an accurate check of the request legitimacy and of the correct election closure, officially delivers the private key K_E^- to the election supervisor. Furthermore, since each voting packet was digitally signed by the virtual ballot box before being saved, the system checks, for each vote, that the packet has not been modified and that the signature is still valid (Figure 7-a), before actually counting it.

The procedure used by the system to decrypt each voting packet is explained in Figures 7-b and 7-c. Please recall that voting packets were encrypted with symmetric keys, K_i, one for each voting packet. Then, each K_i was encrypted with the current election's public key and stored in the XML file, along with the actual vote.

During the tallying phase, each symmetric key K_i is decrypted by using the private key provided by the notary, and then it is used to decrypt the corresponding voting packet. As shown in Figure 7-d, decrypted voting packages are used to update the vote count, based on the preferences specified on the voting card. To ensure the integrity and authenticity of the entire counting process, then, final results are digitally signed by the election supervisor.

6. Formal Verification of SecureBallot

In this section we present the formal security analysis of the SecureBallot protocol. Since the security of cryptographic protocols cannot be reasonably verified manually, we adopted an automatic tool which was proven to be able to uncover large number of attacks against many security protocols that were previously thought to be secure (Cremers et al., 2009).

Some of the most widely used techniques for automated modeling and analysis of security protocols exploit tools like CSP (Communicating Sequential Processes) and FDR (Failure-Driven Refinement). CSP is a formal language originally proposed by Hoare (1978) for describing interactions between two or more processes, at any level of abstraction, while FDR is a model checking tool (Gibson-Robinson et al., 2014).

FDR requires as input the system specification and implementation of a protocol written in CSP and produces a counterexample, if the protocol is vulnerable to some kind of attack. FDR uses the state space search technique to find any sequence of messages that may lead to an attack on the analyzed protocol. CSP and FDR have been extensively used for discovering attacks in many protocols (Xu et al., 2008; Raju and Kumari, 2011). Notoriously, Lowe (1996) used this approach to find an attack against Needham-Schroeder authentication protocol, fixing it.

Unfortunately, describing security protocols in CSP is a long and laborious process, which can inevitably lead to modeling errors. For this reason, a compiler called Casper was proposed by Lowe (1997). Casper starts from a high-level description of the security protocol and automatically produces the corresponding CSP code. FDR is then used to analyze the protocol and verify its security properties.

In the following, we present the modeling of the SecureBallot protocol exploiting Casper and the resulting analysis performed with FDR.

We will present only relevant sections of the protocol description, and defer to Appendix A for the complete code of the model. Casper input files are divided into several sections; the most significant ones for understanding how protocol verification is performed are five: `#Free variables`, `#Protocol description`, `#Processes`, `#Specification` and `#Intruder information`.

In the `#Free variables` section, we define the types of variables and functions used in the protocol, as shown in the following excerpt:

```
#Free variables
V : VStation
B : BBox
S : ESup

ni : Nonce
vi : Vote
ivi : InitializationVector
ki : SymmetricKey
kvb : SessionKey
h: HashFunction

PK : ESup -> PublicKey
SK : ESup -> SecretKey
bbpk : PublicKey
bbsk : PrivateKey
```

where `V`, `B` and `S` are, respectively, a voting station, the virtual ballot box and the election supervisor, as described in previous sections. Other variables' definitions follow the same nomenclature presented in Table 2. `PK` and `SK` are functions that return the public and private keys of the election, respectively. `PK` is a function known by all agents participating in the protocol (potentially also by attackers), while `SK` is computable only by the election supervisor (note that in SecureBallot this private key is disclosed only by a notary during the tallying phase). The two variables `bbpk` and `bbsk` constitute the public/private key pair of the virtual ballot box.

The first value is known by the ballot box itself and by the election supervisor, while the private key is known only by the ballot box and will be used to digitally sign the encrypted voting packets.

The `#Processes` section defines the roles of agents in the protocol and their initial knowledge. In our model, the voting station takes the role of `INITIATOR` of the protocol,

the virtual ballot box acts as RESPONDER, and the election supervisor has the role of ESUPERVISOR.

```
#Processes
INITIATOR(V, vi, ni, ki, ivi, kvb, S) knows PK \
    generates ni, ki, ivi
RESPONDER(B, kvb, bbpk, bbsk, S) knows PK
ESUPERVISOR(S, B, bbpk) knows PK, SK(S)
```

The #Protocol description section contains all the messages exchanged by the various agents, with a notation similar to that commonly used for describing security protocols.

In the protocol description we have included the whole life cycle of a single vote, from when it is encrypted and sent to the virtual ballot box, which accepts it and saves it in the database, until its decryption and counting during the tallying phase.

```
#Protocol description
0.    -> V : B
1a. V -> B : {V, ni}{kvb}, {ki}{PK(S)} % kic, \
             {ivi}{PK(S)} % ivic, {vi}{ki} % vic
1b. V -> B : h(kvb, {ki}{PK(S)} % kic, \
             {ivi}{PK(S)} % ivic, {vi}{ki} % vic, ni)
-- After saving vote into DB
2.  B -> V : {ni}{kvb}
-- Vote tallying phase
3a. B -> S : kic % {ki}{PK(S)}, ivic % {ivi}{PK(S)}, \
             vic % {vi}{ki}, ni
3b. B -> S : {h(vic % {vi}{ki}, kic % {ki}{PK(S)}, \
             ivic %  {ivi}{PK(S)}, ni)}{bbsk}
4.  S -> B : {ni}{SK(S)}
```

Message 0 is used to start the protocol run. It is assumed that V receives an initial message from the *environment*, telling it to run the protocol with B. In Casper, {m}{k} denotes that message m is encrypted with key k. Furthermore, m % v indicates that the receiver of the message should not try to decipher the message m, but should simply store it in a variable v, and then forward it to another agent, as shown in message 1b. Conversely, v % m indicates that the sender submits the contents of the previously defined variable v, which should be deciphered and interpreted by the receiver as the message m, as shown in message 3a. Following the suggestion of the authors of Casper, we split long messages into two parts (e.g., 1a and 1b) in order to reduce the state space explored by FDR during the analysis. This does not alter the protocol properties and the attacks identified by Casper/FDR.

Message 1a contains the identity of the voting station (V) and the nonce ni, both encrypted with the session key known only to V and B, the symmetric key ki and the IV ivi, both encrypted with the public key of the election, and the vote encrypted with ki. Message 1b contains the HMAC of the vote, IV, and ki key (all encrypted), computed with the session key kvb. Note that the virtual ballot box cannot decrypt the vote, the IV and the symmetric key; this is indicated by the %-notation, as stated above. If the protocol succeeded up to this point, the encrypted vote is digitally signed and saved in the database. Afterwards, the virtual ballot box sends message 2, which is an acknowledgment. The fact that message 2 is encrypted with kvb assures V of B's identity.

After the election is over, during the tallying phase, all encrypted votes are sent to the election supervisor.

In messages 3a and 3b the supervisor receives the encrypted vote and its digital signature and can verify it using the public key of the virtual ballot box, bbpk. At this point the supervisor has the necessary knowledge to decrypt the vote and count it correctly. Finally, message 4 is an acknowledgement that the previous messages have been received and the protocol has been successfully completed. Note that, in our protocol description, we show the tallying of a single vote vi.

To execute the protocol, Casper needs to instantiate "actual variables" corresponding to the roles defined in the #Free variables section. In SecureBallot, Alice is a voting station (V), Bob is the virtual ballot box (B), Susan is the election supervisor (S) and Mallory is the intruder with malicious intentions.

The security requirements of the protocol are defined in the #Specification section:

```
#Specification
StrongSecret(V, vi, [S])    -- S1
StrongSecret(V, ki, [S])    -- S2
StrongSecret(V, ivi, [S])   -- S3

Agreement(B, V, [ni])       -- A1
Agreement(V, B, [ni])       -- A2
Agreement(B, S, [ni])       -- A3
Agreement(S, B, [ni])       -- A4

-- Custom specification     -- C1
if Bob receives message 4 from Susan then previously
    Alice sends message 1a to Bob containing vi
```

The Secret type specification indicates which variables should remain secret. In our case, the vote, symmetric key, and IV must remain secret from everyone except V and S. Specifically, StrongSecret indicates that the values must remain secret even in the case of an incomplete run. In SecureBallot, it is obviously critical that the vote remains secret at all costs, to preserve user privacy.

Authentication requirements are specified with the Agreement type specification. We want to ensure that, at the end of the protocol, V is mutually authenticated with B and the latter is mutually authenticated with S. Trying to add an additional Agreement specification between V and S, Casper reports that this is not possible because "*S is not causally linked to V*". This basically confirms that the vote is completely unrelated to the voter and to the voting station it was sent from.

Nevertheless, we added an additional custom specification to indicate that, if Susan sends the final acknowledgment to Bob (message 4), then the vote that is being counted must have been previously sent by an authorized voting station (i.e., by Alice and not by the intruder Mallory).

Finally, the #Intruder information section is used to specify the identity and initial knowledge of the intruder:

Table 3: Security evaluation of SecureBallot, as described in the `#Specification` section of the Casper model.

	S1	S2	S3	A1	A2	A3	A4	C1
SecureBallot without TLS	✓	✓	✓	✓	✓	✓	✓	✓
SecureBallot	✓	✓	✓	✓	✓	✓	✓	✓

Table 4: Security evaluation of SecureBallot in the unrealistic case where the session key `kvb` is compromise.

	S1	S2	S3	A1	A2	A3	A4	C1
SecureBallot without TLS	✓	✓	✓	-	-	✓	✓	-
SecureBallot	✓	✓	✓	✓	✓	✓	✓	✓

```
#Intruder information
Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Susan, Nm, Vm,
                     Ivm, Km, PK}
```

In our formal description, the intruder `Mallory` knows the identities of `Alice`, `Bob`, and `Susan` in advance. In addition, she can obtain the public key of other agents (using `PK` function) and create forged versions of vote, symmetric key, IV and nonce. `Mallory`'s goal is to break one or more of the specifications.

Analyzing the output that Casper generates from our model, FDR does not detect any sequence of messages that violates the security requirements of SecureBallot. Note that we have not considered an important security feature of SecureBallot, namely that all messages are exchanged via a secure and authenticated communication channel, exploiting TLS. Even without this important feature, FDR does not detect any attack against our protocol. This proves the robustness of SecureBallot, despite the lack of security of the communication channel used.

In the unlikely event that the intruder learns the session key `kvb`, some messages could be forged, violating the authentication specifications. In particular, if we add `kvb` to `Mallory`'s knowledge, FDR finds the following attack:

```
1a. Alice -> I_Bob   : {Alice, Ni}{KVB}, {Ki}{PK(Susan)},
                       {Ivi}{PK(Susan)}, {Vi}{Ki}
1b. Alice -> I_Bob   : h(KVB, {Ki}{PK(Susan)},
                       {Ivi}{PK(Susan)}, {Vi}{Ki}, Ni)
2.  I_Bob   -> Alice : {Ni}{KVB}
```

Here the intruder pretends to be `Bob` and manages to deceive `Alice`, leading the voting station to believe that the submission protocol has been correctly completed with the real `Bob`, although this is not the case.

Additionally, the knowledge of this session key allows the intruder to perform another elaborate attack, which we report below:

```
1a. I_Alice -> Bob   : {Alice, Nm}{KVB}, {Km}{PK(Susan)},
                       {Ivm}{PK(Susan)}, {Vm}{Km}
1b. I_Alice -> Bob   : h(KVB, {Km}{PK(Susan)},
                       {Ivm}{PK(Susan)}, {Vm}{Km}, Nm)
2.  Bob   -> I_Alice : {Nm}{KVB}
3a. I_Bob   -> Susan : {Km}{PK(Susan)},
                       {Ivm}{PK(Susan)}, {Vm}{Km}, Nm
3a. Bob     -> I_Susan : {Km}{PK(Susan)}, {Ivm}{PK(Susan)},
                       {Vm}{Km}, Nm
3b. Bob     -> I_Susan : {h({Vm}{Km}, {Km}{PK(Susan)},
                       {Ivm}{PK(Susan)}, Nm)}{BBsk}
3b. I_Bob   -> Susan : {h({Vm}{Km}, {Km}{PK(Susan)},
                       {Ivm}{PK(Susan)}, Nm)}{BBsk}
4.  Susan -> I_Bob   : {Nm}{SK(Susan)}
4.  I_Susan -> Bob   : {Nm}{SK(Susan)}
```

Please note that the possibility that the intruder is in possession of this key is very unlikely, unless she manages to tamper with one of the voting stations, which are constantly surveilled by the electoral staff.

Even in this unrealistic case, however, the intruder cannot violate the `StrongSecret` specifications, so voter privacy is still preserved. Moreover, as noted above, SecureBallot uses TLS for all communications. We can model this fact in Casper by adding the security features of the communication channel:

```
#Channels
secret
authenticated
```

Taking into account the secrecy and authentication properties of the communication channel, FDR no longer finds any attack against SecureBallot, even if the session key `kvb` is compromised.

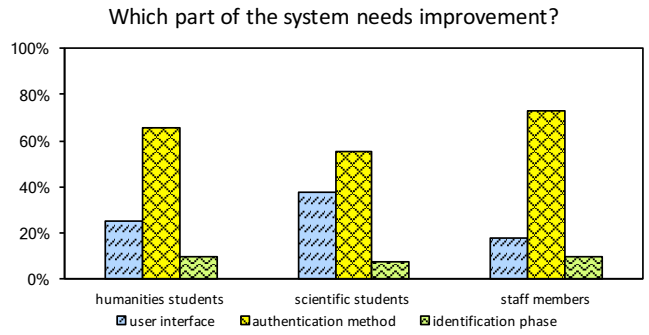
Tables 3 and 4 summarize our evaluation of SecureBallot's security specifications. Notably, Table 3 shows that SecureBallot's protocol is always secure, regardless of the communication channel used. Even in the extreme case where an intruder managed to tamper with a voting station and obtain the `kvb` key, TLS would allow SecureBallot to keep all its security properties intact, as shown in Table 4.

7. Case study

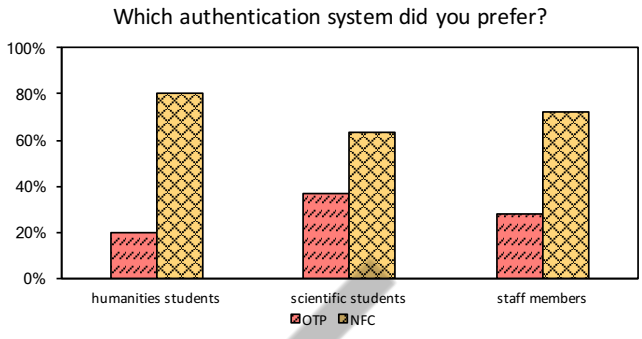
To test SecureBallot in increasingly challenging conditions, we have chosen a case study that is well suited to demonstrate the viability of the system. The characteristics we looked for concerned the number and heterogeneity of potential users, the frequency of elections of different scale and importance, as well as the prior familiarity of voters with IT tools.

Considering all these aspects together, university elections seem like the ideal environment to test SecureBallot. Different elections are constantly taking place within universities, for example to renew student and faculty representatives, with some of them (especially those involving students) attracting a large number of voters.

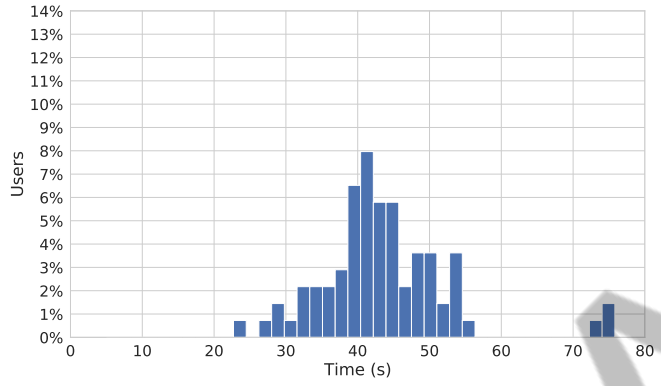
Moreover, university staff are generally accustomed to using IT tools, so they are well disposed towards technological innovations in various areas, including elections. Likewise, students are mostly digital natives and are used to interact with technological tools to attend lessons or take exams.



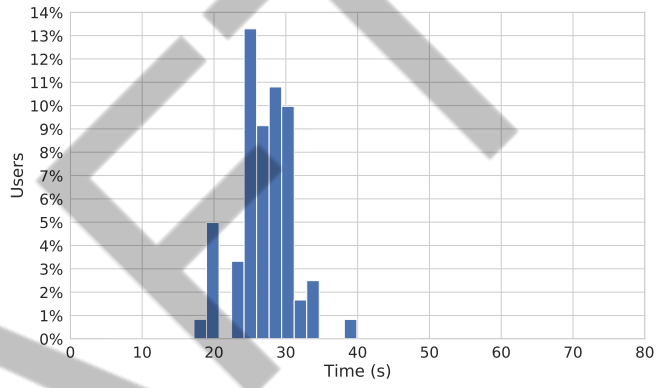
(a) Question about possible improvements.



(b) Satisfaction comparison between OTP and NFC.



(c) Time spent by users inside the voting booth, from the first interaction with the old OTP-based system to the completion of the voting operation.



(d) Time spent by users inside the voting booth, from the first interaction with the new NFC-based system to the completion of the voting operation.

Figure 8: Tests performed during the transition from a preliminary version of SecureBallot that used OTP tokens to the final version that uses NFC.

Traditionally, elections held at the University of Palermo did not rely on IT technologies. Public polling stations were located in different areas of the university campus for voting purposes, but electors could only express their preferences in the polling station assigned to them. This prevented a person from showing up at multiple polling stations to vote more than once, but it also greatly reduced the convenience and efficiency of voting operations.

The polling and voting stations were supervised by staff members who identified voters and ensured the fairness of elections. This entailed the involvement of regular staff members, who had to temporarily suspend their usual work activities during elections, slowing down other tasks.

After the identification phase, each voter was given an indelible pencil and a paper ballot card, which were created and printed specifically for that election. Voters then marked the paper ballot card, indicating their preferences, and returned it to staff members. The voting operation ended by entering the ballot card in a special ballot box. Then, in the tallying phase, voting cards were counted manually. The final results were usually published several hours after the end of the election.

SecureBallot does not have these limits: voters can freely choose which polling station to vote at, and election results are computed efficiently in a secure manner. Furthermore, many universities already possess the ICT infrastructure required to operate SecureBallot, i.e. reliable Internet connections (both wireless and wired) and PCs that can be easily used as voting stations.

In the proposed case study, we used off-the-shelf laptop computers and secure contactless smartcards with NFC technology as unlocking tokens to enable voting stations.

Each polling station is equipped with an NFC writer and several readers, one per voting booth. The voter, once authorized, receives one of the NFC tags with a new code written by the proper NFC writer. This tag will only enable the voting station indicated by SecureBallot.

When the user has finished voting, he returns the NFC tag, which is then overwritten by staff members using the appropriate NFC writer. The same authorization process can be achieved by means of other kinds of devices, such as OTP tokens.

7.1. System evaluation

During a period of six months, several tests have been carried out at the University of Palermo, both on mock

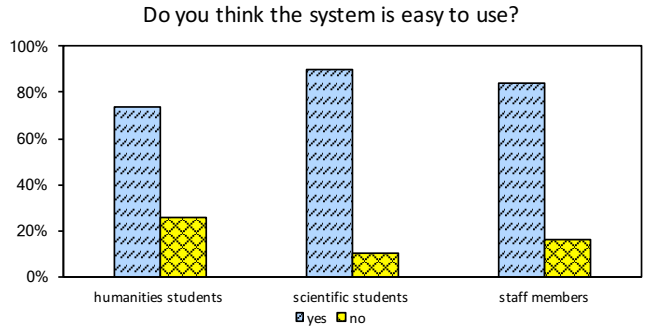


Figure 9: Question about ease of use.

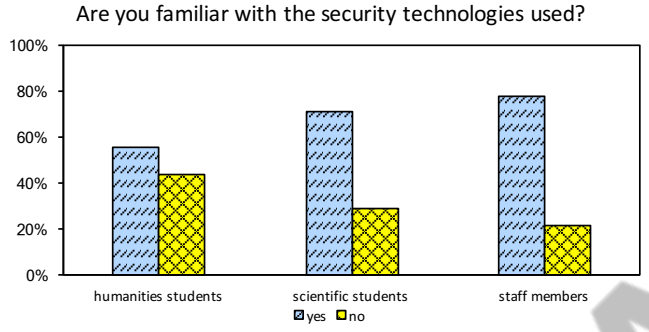


Figure 10: Question about familiarity with security technologies.

elections and on real ones, to perform a thorough evaluation of SecureBallot and to repeatedly improve it based on users' feedback. All tests were performed on off-the-shelf laptops equipped with Intel 3805U 1.9GHz CPU and 4GB RAM. Several mock elections were conducted with fictional candidates, employing volunteer students and university staff as voters. SecureBallot has been tested in all its components in increasingly challenging scenarios, including hundreds of voters and multiple polling stations spread across the campus. After each voting procedure, participants were given a questionnaire similar to those proposed in numerous works in the literature in the field of e-Voting systems (Ochoa and Peláez, 2017; Pomares et al., 2014). The questionnaires were designed to ask voters for their satisfaction in using the system and to encourage them to make suggestions for improving SecureBallot.

Most of the feedbacks and requests from both staff and voters were related to user interface refinements and bug fixes, which we have addressed between successive versions of the system. Indeed, we have noticed a constant increase in user satisfaction between versions.

The first mock elections were based on a preliminary version of SecureBallot, which exploited an OTP token generator for enabling the voting station, instead of NFC tags. In particular, in order to enable the voting station, participants had to use the token OTP hardware, pressing a button and entering the random sequence of numbers generated.

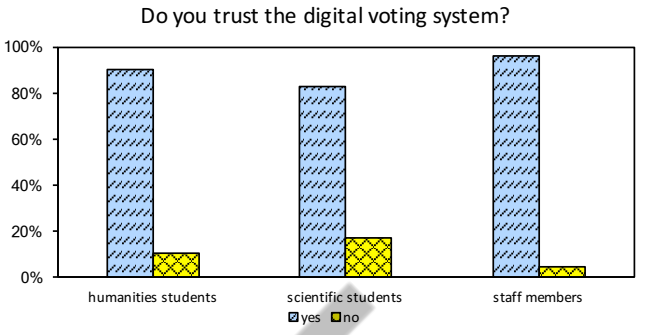


Figure 11: Question about trust in the e-Voting system.

To check if OTP-based interaction was actually perceived as laborious by users, we asked them what part of the system they felt needed improvements. For statistical purposes, we asked students about their educational background, i.e. whether their studies related to scientific or humanities areas. As Figure 8a shows, most users, regardless of the category they belonged to, replied that the main element to improve was the OTP-based authentication. The percentage of users who reported OTP as a problem is higher in the staff members category, which can be explained by their higher average age compared to students.

In order to understand the problems presented by OTP authentication, we asked, through an open question, what was the main issue. The answers we obtained highlighted problems related to the digits that were too small to read on the OTP token display and the difficulty to input them using the voting station keyboard in a short time. This was compounded by the time-based nature of OTP authentication, which meant that codes would expire before some users could type them.

Moreover, in order to measure the actual usability and ease of use of the system with OTP token generator, we decided to measure the turnaround time, defined as the time elapsed from the moment the user started to interact with the voting station, until the completion of the voting operation.

As Figure 8c shows, the time taken by users with the first version of the system roughly follows a Gaussian distribution, with an average of 43.57 seconds. Although most users completed voting operations within a reasonable timeframe, some voters, being less familiar with digital technologies, took a relatively long time (almost 80 seconds) and required additional reassurances and explanations from staff members.

Given these results, we adopted a new authentication method based on NFC tags, which avoids all the problems identified since it is not time-based and does not require small digits to type in.

Using a contactless protocol to transfer a sequence of data, an NFC tag is equipped with a microchip and an antenna, and it can be easily read and re-written simply

Table 5: Real elections setup.

Case study	First	Second	Third
Entitled to vote	416	127	217
Attendance	71%	70%	73%
# voting cards	5	5	15
Multiple preferences	-	✓	✓
Total preferences	297	116	297

Table 6: Traditional university voting vs. SecureBallot.

	Traditional	SecureBallot
Setup time	2-3 days	1-2 hours
Paper wasted	Proportional to voters	Negligible amount
Flexibility	Polling station assigned to voters	Voters can choose any polling station
Spoilt votes	About 3%	None
Tallying time	Proportional to voters (several hours)	Proportional to voters (milliseconds)

by holding the tag close to compatible NFC readers/writers. Users can then enable a voting station by holding the proper NFC tag near a special reader within the booth.

This solution reduced the time spent by users inside the voting booth by about 18 seconds (40% less), on the average, as Figure 8d shows. Moreover, as can be inferred from the comparison between Figures 8c and 8d, the variance of the time needed for voting was also much smaller with the new system, which means that authentication via NFC tag is used with similar results by all voters, levelling out the differences between various types of users.

Finally, to confirm the effectiveness of the new approach, we asked users which of the two authentication systems they preferred, between the OTP-based and the NFC-based one. As Figure 8b shows, all categories of users confirmed that they preferred the new system based on NFC tags.

Figure 9 shows the results collected during the last mock election. The questionnaire revealed that about 90% of students with a scientific background considered SecureBallot easy to use, as shown in Figure 9. Among humanities students, the percentage is slightly lower, with about 75% positive feedback.

This difference is probably due to the greater familiarity of the former with new technologies. Indeed, comparing these results with the answers to the second question (Figure 10), it is possible to notice a correlation between voters' familiarity with the adopted technologies and the perceived ease of use.

University staff members work daily using security technologies similar to those employed in SecureBallot, such as digital signature and weak authentication, as demonstrated in Figure 10. Unsurprisingly, then, Figure 9 shows that about 84% of staff members expressed a favorable opinion on SecureBallot.

The third question was related to voters' trust in the use of SecureBallot. In this regard we have obtained the most surprising results: humanities students, who are less familiar with the technologies used by SecureBallot, have a greater confidence in the new voting system than their colleagues with a scientific background, as shown in Figure 11. Nevertheless, we can notice that about 90% of voters, regardless of their studies, trust SecureBallot.

After receiving positive feedback from mock elections tests, we started to use SecureBallot for real elections at the University of Palermo. Table 5 summarizes voters' affluence and other relevant statistics of three real-world tests we have carried out. We chose to present these particular elections as they highlight specific challenges, and have allowed us to test SecureBallot in increasingly difficult conditions.

In the first real election test, SecureBallot was used by 297 participants (out of 416 entitled to vote), who belonged to 5 different categories of voters depending on their role within the university. The test took place in the Engineering Department of the University of Palermo. Each user was given the opportunity to vote on a single ballot pertaining to his category, choosing one among several candidates. University staff members were employed as election officials and multiple polling stations were used, so that users could decide where to vote.

In the second test, the number of voters was actually lower than in the first one, due to the number of people entitled to vote, which was also lower (127 vs. 416, respectively). However, the second test was also challenging, because electors could fill out multiple voting cards. Indeed, it often happens that multiple elections for the same community are grouped on a single day for organizational reasons and voters are asked to cast multiple voting cards (one for each type of election). In our second test, people voted to elect representatives from many university degree courses. Each person had the right to vote for representatives of their own course. However, some users were affiliated with more than one course and they were therefore able to cast (a single time) multiple voting cards simultaneously. This type of election was therefore more complex from an organizational point of view, but SecureBallot efficiently handled this increased complexity.

The third test involved 217 people entitled to vote, which could also cast multiple voting cards. In this test, however, the number of different voting cards was much higher (15 vs. 5 in the previous cases). Plus, depending on electors' categories, some users could fill out only 1 or 2 voting cards, while others could fill out up to 5 different ones. This meant that the possible "voter-voting card combinations" were significantly more than in the first two cases, and required an improved user interface of the soft-

ware used by staff members in the preparatory phase. Voters in the real elections filled in questionnaires similar to those used in the mock elections. The results obtained are in line with those collected previously, which proves the validity of the system.

Finally, Table 6 presents some considerations on the use of SecureBallot compared to the corresponding traditional voting system.

In particular, during the preparatory phase, setup time is drastically reduced in SecureBallot (a couple hours vs. multiple days), and e-Voting systems in general are obviously significantly more eco-friendly in terms of paper waste.

In the voting phase, SecureBallot's flexibility allows users to choose the polling station where to vote, unlike traditional systems. As mentioned, SecureBallot also ensures that all votes cast by users are valid. On the contrary, in our university, we have recorded about 3% of spoilt votes in past elections that exploited a paper-based voting system.

Finally, the time needed to count the votes is proportional to the number of voters, both in traditional systems and in SecureBallot. However, the tallying phase in SecureBallot only lasts a few milliseconds, while in traditional systems it could take several hours, or even days.

8. Discussion

SecureBallot exploits well known cryptographic techniques (e.g., RSA and AES) and secure protocols such as TLS. These techniques have been chosen because they represent the state-of-the-art in the cryptography field, and ensure user privacy and voting secrecy, in addition to all the other security properties discussed in Section 4. In this section we present a discussion of other aspects of SecureBallot, and specifically those related to scalability.

Indeed, scalability is a critical aspect of an e-Voting system, given that, potentially, millions of voters can participate in a single election. From an organizational standpoint, e-Voting systems are at an advantage over traditional ones. As the electoral pool increases, the benefits of a non-paper-based system become more obvious (e.g., not having to print and distribute millions of paper ballots).

Computational efficiency is a different matter altogether. The use of computationally expensive cryptographic techniques could greatly reduce the efficiency of an e-Voting system and, in more severe cases, even cause it to be unfeasible for elections with a large user base. This aspect is even more prominent when considering remote e-Voting, in which heterogenous users' devices (e.g., smartphones, tablets, outdated PCs) are demanded to perform some processing tasks.

SecureBallot is a supervised system and the election organizers can choose in advance the devices to employ as voting and polling stations, as well as the hardware components needed to support the backend operations. Nevertheless, in order to understand the requirements of such devices, it is interesting to analyze the computational complexity of the most time-consuming operations performed by SecureBallot. This will also let us verify whether our system can be implemented on resource-constrained devices, and if it is indeed scalable when the user base is significantly larger than that of university elections.

Certain tasks, such as the generation of session keys used to calculate the HMAC, are performed only once during the preparatory phase of the election and therefore do not impact the efficiency of voting operations. As regards the encryption of each vote, which is performed by voting stations, the most costly cryptographic technique involved is asymmetric encryption, performed with RSA. However, in SecureBallot we don't employ RSA to encrypt the whole vote, but rather to encrypt the AES symmetric key that will be used as KEK (key encryption key). Since the length of the AES key is fixed, the amount of data encrypted with RSA for each vote does not depend on the number of users, so that such encryption operation is performed in constant time.

Likewise, the size of the vote packet that is encrypted with AES and authenticated with HMAC is also essentially fixed, as it can vary negligibly based only on the characteristics of the specific election.

Afterwards, the virtual ballot box verifies the HMAC of a fixed size voting packet and digitally signs it if it is accepted. Thus, the computational complexity of all cryptographic operations performed during the voting phase is constant because it does not depend in any way on the number of users and votes cast.

During the tallying phase, the operations carried out to decrypt individual votes (i.e., verifying the digital signature, decrypting the KEK with RSA, and decrypting the actual vote with AES) also exhibit constant complexity because they are all performed on data blocks of fixed size. Since all votes must be decrypted, the total complexity of the tallying phase will thus be proportional to the number of votes cast.

Technically, tallying is the only operation that is not performed in constant time, but it is worth noticing that decryption is performed by back-end devices, which can be chosen in advance depending on the scale of the election. From our tests, the entire tallying phase is completed in milliseconds, even when using relatively dated hardware. If necessary, however, the tallying operations could be easily parallelized, since the decryption of a single vote is obviously independent from that of the others.

As shown in Figure 8d, the average time it takes users to cast a single vote is about 30 seconds. On the other hand, the time it takes a voting station to encrypt a single vote is on the order of a few milliseconds on the laptops we used in the case study. The encryption time is negligible compared to the total voting time, which means that SecureBallot could easily operate even on low-cost PCs or other resource-constrained devices. Ultimately, from a cryptographic perspective, SecureBallot is highly scalable and perfectly usable even for large elections.

Given that the computational complexity of the cryptographic operations carried out by SecureBallot is negligible, compared to the total time needed to cast a vote, in order to estimate the overall scalability of the system it is worth discussing organizational aspects as well. Obviously, organizing a nationwide election, with millions of voters, is quite different from organizing a university election. The number of voting and polling stations must be carefully chosen by the administrators to avoid long queues. Based on our experience, voter registration, which generally involves ID verification, is often the most time-consuming step, and therefore we suggest using an appropriate number of polling stations for the expected voter turnout.

9. Conclusions

In this work we proposed SecureBallot, a new electronic voting system suitable to any supervised election and extensively tested it in the scenario of university elections. Given the great variability of such elections, it was possible to perform multiple tests that highlighted the high user satisfaction of voters that used SecureBallot.

SecureBallot meets all the security properties necessary to ensure the privacy of users and the fairness of elections. To this end, the identification and voting phases are completely decoupled, and no information about users is collected. The use of state-of-the-art encryption techniques guarantees the secrecy of votes. The whole protocol has been formally verified by using Casper/FDR, demonstrating that SecureBallot is always secure, regardless of the communication channel used.

One of the main drawbacks of traditional elections is undoubtedly the high operating costs. Adopting SecureBallot can make electoral procedures more cost effective, because the costs incurred initially can be amortized over the years in numerous elections.

In addition to making electoral procedures more convenient in terms of resources and personnel involved, saving time and money, the adoption of SecureBallot also improve vote counting, which is one of the most critical phases of all elections, with a substantial time gain. Automating this process has the added benefit of avoiding inconsistencies typical of tallying phases in traditional voting systems. Finally, our tests demonstrated that using technologies familiar to voters greatly increases user satisfaction and trust in digital voting systems, and consequently in electronic elections themselves.

Ultimately, the feasibility of using electronic voting systems is still debated around the world today. SecureBallot aims to gain the trust of voters and be used in elections of all complexities. To this end, SecureBallot guarantees strict security properties such as incoercibility, which is critical in most elections. Moreover, since SecureBallot is totally open-source, security experts from all over the world can analyze its source code and verify its robustness.

Appendix A. Formal verification of SecureBallot

In this section, we present the full code of the Casper model, which can be used to reproduce the tests of the formal verification described in Section 6.

```
#Free variables
V : VStation
B : BBox
S : ESup

ni : Nonce
vi : Vote
ivi : InitializationVector
ki : SymmetricKey
kvb : SessionKey
h: HashFunction

PK : ESup -> PublicKey
SK : ESup -> SecretKey
bbpk : PublicKey
bbsk : PrivateKey

InverseKeys = (PK, SK), (kvb, kvb), (ki, ki), (bbpk, bbsk)

#Processes
INITIATOR(V, vi, ni, ki, ivi, kvb, S) knows PK generates ni
     , ki, ivi
RESPONDER(B, kvb, bbpk, bbsk, S) knows PK
ESUPERVISOR(S, B, bbpk) knows PK, SK(S)

#Protocol description
0.    -> V : B
1a. V -> B : {V, ni}{kvb}, {ki}{PK(S)} % kic, \
             {ivi}{PK(S)} % ivic, {vi}{ki} % vic
1b. V -> B : h(kvb, {ki}{PK(S)} %kic, \
             {ivi}{PK(S)} % ivic, {vi}{ki} % vic, ni)
-- Save vote into DB
2.  B -> V : {ni}{kvb}
-- Vote tallying
3a. B -> S : kic % {ki}{PK(S)}, ivic % {ivi}{PK(S)}, \
             vic % {vi}{ki}, ni
3b. B -> S : {h(vic % {vi}{ki}, kic % {ki}{PK(S)}, \
             ivic %  {ivi}{PK(S)}, ni)}{bbsk}
4.  S -> B : {ni}{SK(S)}

#Specification
StrongSecret(V, vi, [S])
StrongSecret(V, ki, [S])
StrongSecret(V, ivi, [S])

Agreement(B, V, [ni])
Agreement(V, B, [ni])
Agreement(B, S, [ni])
Agreement(S, B, [ni])

if Susan receives message 3a from Bob containing Vi for vi
     then previously Alice sends message 1a to Bob
     containing Vi for vi

#Actual variables
Alice, Mallory : VStation
Susan : ESup
Bob : BBox
Ni, Nm : Nonce
Vi, Vm : Vote
Ivi, Ivm : InitializationVector
Ki, Km : SymmetricKey
KVB : SessionKey
BBpk : PublicKey
```

```
BBsk : PrivateKey

InverseKeys = (KVB, KVB), (Ki, Ki), (Km, Km), (BBpk, BBsk)

#Functions
symbolic PK, SK

#System
INITIATOR(Alice, Vi, Ni, Ki, Ivi, KVB, Susan)
RESPONDER(Bob, KVB, BBpk, BBsk, Susan)
ESUPERVISOR(Susan, Bob, BBpk)

#Intruder information
Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Susan, Nm, Vm, Ivm, Km, PK
    }
```

References

Adida, B., 2008. Helios: Web-based open-audit voting., in: USENIX security symposium, pp. 335–348.

Agarwal, H., Pandey, G.N., 2014. A secure e-election system, in: 2014 International Conference on Information Science Applications (ICISA), pp. 1–4. doi:10.1109/ICISA.2014.6847335.

Agate, V., Curaba, M., Ferraro, P., Lo Re, G., Morana, M., 2020. Secure e-voting in smart communities, in: ITASEC 2020, pp. 1–11.

Aggarwal, S., Chaudhary, R., Aujla, G.S., Kumar, N., Choo, K.K.R., Zomaya, A.Y., 2019. Blockchain for smart communities: Applications, challenges and opportunities. Journal of Network and Computer Applications 144, 13 – 48. URL: http://www.sciencedirect.com/science/article/pii/S1084804519302231, doi:https://doi.org/10.1016/j.jnca.2019.06.018.

Ahmad, T., Hu, J., Han, S., 2009. An efficient mobile voting system security scheme based on elliptic curve cryptography, in: 2009 Third International Conference on Network and System Security, pp. 474–479. doi:10.1109/NSS.2009.57.

Alvarez, R.M., Katz, G., Pomares, J., 2011. The impact of new technologies on voter confidence in latin america: Evidence from e-voting experiments in argentina and colombia. Journal of Information Technology & Politics 8, 199–217.

Antoniou, A., Korakas, C., Manolopoulos, C., Panagiotaki, A., Sofotassios, D., Spirakis, P., Stamatiou, Y.C., 2007. A trust-centered approach for building e-voting systems, in: International Conference on Electronic Government, Springer. pp. 366–377.

Avgerou, C., Masiero, S., Poulymenakou, A., 2019. Trusting e-voting amid experiences of electoral malpractice: The case of indian elections. Journal of Information Technology .

Bederson, B.B., Lee, B., Sherman, R.M., Herrnson, P.S., Niemi, R.G., 2003. Electronic voting system usability issues, in: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 145–152.

Bélanger, F., Carter, L., 2008. Trust and risk in e-government adoption. The Journal of Strategic Information Systems 17, 165–176.

Bohli, J.M., Müller-Quade, J., Röhrich, S., 2007. Bingo voting: Secure and coercion-free voting using a trusted random number generator, in: International Conference on E-Voting and Identity, Springer. pp. 111–124.

Carroll, T.E., Grosu, D., 2009. A secure and anonymous voter-controlled election scheme. Journal of Network and Computer Applications 32, 599 – 606. URL: http://www.sciencedirect.com/science/article/pii/S1084804508000830, doi:https://doi.org/10.1016/j.jnca.2008.07.010.

Carter, L., Bélanger, F., 2005. The utilization of e-government services: citizen trust, innovation and acceptance factors. Information systems journal 15, 5–25.

Clarkson, M.R., Chong, S., Myers, A.C., 2008. Civitas: Toward a secure voting system, in: 2008 IEEE Symposium on Security and Privacy (sp 2008), pp. 354–368.

Cremers, C.J., Lafourcade, P., Nadeau, P., 2009. Comparing state spaces in automatic security protocol analysis, in: Formal to Practical Security. Springer, pp. 70–94.

Falleti, T.G., Lynch, J.F., 2009. Context and causal mechanisms in political analysis. Comparative political studies 42, 1143–1166.

Frankland, R., Demirel, D., Budurushi, J., Volkamer, M., 2011. Side-channels and evoting machine security: Identifying vulnerabilities and defining requirements, in: 2011 International Workshop on Requirements Engineering for Electronic Voting Systems, pp. 37–46. doi:10.1109/REVOTE.2011.6045914.

Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W., 2014. Fdr3 — a modern refinement checker for csp, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer. pp. 187–201.

Gritzalis, D.A., 2002. Principles and requirements for a secure e-voting system. Computers & Security 21, 539–556.

Hall, J.L., 2006. Design and the support of transparency in vvpat systems in the us voting systems market. UC Berkeley, School of Information .

Hanifatunnisa, R., Rahardjo, B., 2017. Blockchain based e-voting recording system design, in: 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), IEEE. pp. 1–6.

Hjálmarsson, F.Þ., Hreiðarsson, G.K., Hamdaqa, M., Hjálmtýsson, G., 2018. Blockchain-based e-voting system, in: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), IEEE. pp. 983–986.

Hoare, C.A.R., 1978. Communicating sequential processes. Communications of the ACM 21, 666–677.

Khamitov, I.M., Dostov, V., Shoust, P., 2019. Secret voting: Knowledge vs trust, in: International Conference on Computational Science and Its Applications, Springer. pp. 577–586.

Kohno, T., Stubblefield, A., Rubin, A.D., Wallach, D.S., 2004. Analysis of an electronic voting system, in: IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004, pp. 27–40.

Kshetri, N., Voas, J., 2018. Blockchain-enabled e-voting. IEEE Software 35, 95–99.

Kshetri, N., Voas, J., 2018. Blockchain-enabled e-voting. IEEE Software 35, 95–99. doi:10.1109/MS.2018.2801546.

Küsters, R., Liedtke, J., Müller, J., Rausch, D., Vogt, A., 2020. Ordinos: A verifiable tally-hiding e-voting system, in: 2020 IEEE European Symposium on Security and Privacy (EuroS P), pp. 216–235. doi:10.1109/EuroSP48549.2020.00022.

Lowe, G., 1996. Breaking and fixing the needham-schroeder public-key protocol using fdr, in: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Springer. pp. 147–166.

Lowe, G., 1997. Casper: A compiler for the analysis of security protocols, in: Proceedings 10th Computer Security Foundations Workshop, IEEE. pp. 18–30.

Meier, A., 2012. eDemocracy & eGovernment: Stages of a Democratic Knowledge Society. Springer Berlin Heidelberg.

Moynihan, D.P., 2004. Building secure elections: e-voting, security, and systems theory. Public administration review 64, 515–528.

Mustafa, M.K., Waheed, S., 2020. An e-voting framework with enterprise blockchain, in: Advances in Distributed Computing and Machine Learning. Springer, pp. 135–145.

Neumann, P.G., 1993. Security criteria for electronic voting, in: 16th National Computer Security Conference.

Ochoa, X., Peláez, E., 2017. Affordable and secure electronic voting for university elections: The save case study, in: 2017 Fourth International Conference on eDemocracy eGovernment (ICEDEG), pp. 110–117. doi:10.1109/ICEDEG.2017.7962520.

Park, S., Specter, M., Narula, N., Rivest, R.L., 2021. Going from bad to worse: from internet voting to blockchain voting. Journal of Cybersecurity 7, tyaa025.

Pomares, J., Levin, I., Alvarez, R.M., Mirau, G.L., Ovejero, T., 2014. From piloting to roll-out: voting experience and trust in the first full e-election in argentina, in: 2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE), pp. 1–10. doi:10.1109/EVOTE.2014.7001136.

Raju, K.K., Kumari, V.V., 2011. Formal verification of ieee802. 11i wpa-gpg authentication protocol, in: International Conference on Advances in Information Technology and Mobile Communication, Springer. pp. 267–272.

Rezvani, M., Hamidi, S.M.H., 2010. Mizan: A secure e-voting schema with vote changeability, in: 2010 International Conference on Information Society, pp. 548–552.

Ryan, P.Y., Peacock, T., 2010. A threat analysis of prêt à voter, in: Towards Trustworthy Elections. Springer, pp. 200–215.

Ryan, P.Y.A., Bismark, D., Heather, J., Schneider, S., Xia, Z., 2009. PrÊt À voter: a voter-verifiable voting system. IEEE Transactions on Information Forensics and Security 4, 662–673.

Ryan, P.Y.A., Schneider, S., Teague, V., 2015. End-to-end verifiability in voting systems, from theory to practice. IEEE Security Privacy 13, 59–62.

Sampigethaya, K., Poovendran, R., 2006. A framework and taxonomy for comparison of electronic voting schemes. Computers & Security 25, 137 – 153. doi:https://doi.org/10.1016/j.cose.2005.11.003.

Specter, M.A., Koppel, J., Weitzner, D., 2020. The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in u.s. federal elections, in: 29th USENIX Security Symposium (USENIX Security 20), USENIX Association. pp. 1535–1553. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/specter.

Tonkiss, F., 2009. Trust, confidence and economic crisis. Intereconomics 44, 196–202.

Tornos, J.L., Salazar, J.L., Piles, J.J., 2013. An evoting platform for qoe evaluation, in: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 1346–1351.

Villafiorita, A., Weldemariam, K., Tiella, R., 2009. Development, formal verification, and evaluation of an e-voting system with vvpat. IEEE Transactions on Information Forensics and Security 4, 651–661.

Volkamer, M., McGaley, M., 2007. Requirements and evaluation procedures for evoting, in: Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on, pp. 895–902. doi:10.1109/ARES.2007.124.

Wang, K.H., Mondal, S.K., Chan, K., Xie, X., 2017. A review of contemporary e-voting: Requirements, technology, systems and usability. Data Science and Pattern Recognition 1, 31–47.

Wu, Z.Y., Wu, J.C., Lin, S.C., Wang, C., 2014. An electronic voting mechanism for fighting bribery and coercion. Journal of Network and Computer Applications 40, 139 – 150. URL: http://www.sciencedirect.com/science/article/pii/S1084804513001999, doi:https://doi.org/10.1016/j.jnca.2013.09.011.

Xiao, Z., Xiao, Y., 2012. Security and privacy in cloud computing. IEEE communications surveys & tutorials 15, 843–859.

Xu, S., Huang, C.T., Matthews, M.M., 2008. Modeling and analysis of ieee 802.16 pkm protocols using casperfdr, in: 2008 IEEE International Symposium on Wireless Communication Systems, IEEE. pp. 653–657.

Yao, Y., Murphy, L., 2007. Remote electronic voting systems: an exploration of voters' perceptions and intention to use. European Journal of Information Systems 16, 106–120.

Yi, X., Okamoto, E., 2013. Practical internet voting system. Journal of Network and Computer Applications 36, 378 – 387. URL: http://www.sciencedirect.com/science/article/pii/S108480451200135X, doi:https://doi.org/10.1016/j.jnca.2012.05.005.

Yumeng, F., Liye, T., Fanbao, L., Chong, G., 2012. Electronic voting: A review and taxonomy, in: 2012 International Conference on Industrial Control and Electronics Engineering, pp. 912–917.

Zissis, D., Lekkas, D., 2011. Securing e-government and e-voting with an open cloud computing architecture. Government Information Quarterly 28, 239–251.