PloneVote System Requirements

Lázaro Clapp

23 December 2010

Contents

1 Introduction

This document describes the tasks and responsibilities of the PloneVote system, an extensible software system intended for conducting verifiable elections over the Plone CMS, as well as the requirements that must be fulfilled by its implementation.

In section 2, we describe the problem we seek to address. We discuss the objectives for the PloneVote system for both public and internal use, and provide some definitions related to the problem domain and to some of the concepts used by our solution. In section 3, we talk about the different aspects of the system, centering the discussion around the different concerns the system must deal with. We argue that at least three of the main concerns can be addressed in a roughly orthogonal or independent manner (3.1). We then describe each concern in turn, together with an outline of our proposed solution. These main concerns are: election security (3.2), vote casting and counting (3.3) and election running (3.4).

In section 4 we explore user interaction with the system from the perspective of the different user roles and actions required or permitted by PloneVote, giving a list of detailed use cases that the system should support to be considered complete.

Finally, in section 5, we describe the desired features for the first version of the PloneVote system, categorized by area and each annotated with an implementation priority. The meaning of each priority level is explained in 5.1. Subsection 5.2 comprises the feature listings themselves.

2 Problem description

2.1 Objectives of the system

The PloneVote system aims to deliver a collection of software products - written in python and javascript - which allow users to conduct secure and verifiable elections over the Plone CMS platform. Our goals for the current iteration are twofold:

- We seek to provide a general and extensible framework for conducting various kinds of secure verifiable elections and polls, with different vote casting and vote counting schemes, as well as administrative requirements. To attain this objective, any proposed design must be based in loosely coupled components interacting through clearly defined interfaces and protocols, so that all or most components can be easily substituted to customize election behavior. Components should also be flexible and configurable when possible, so that full replacements are not needed for minor customizations.

- We endeavor to produce a fully working elections package for conducting internal elections within the Institute of Mathematics (IMATE) at UNAM

(Universidad Nacional Autónoma de México, or National Autonomous University of Mexico). The resulting package must be as easy to use as possible, while at the same time providing adecuate security guarantees for its users, such as voter verifiability of the election results and secrecy in vote casting. The package shall make it easy to configure and run an election adhering to the scheduled phases, ballot casting scheme, vote counting scheme, voter authorization, candidate selection and other diverse requirements of common internal elections within IMATE.

The PloneVote prototype to which this document refers is explicitly intended to be used as a platform to research and test known verifiable voting techniques in environments where serious attacks are not expected. We seek to develop a system in which subverting election trust by malicious parties, such as by breaching voter privacy or silently altering the results of an election, is a non-trivial task. However, this is a best-effort research prototype and should not be relied upon for elections that demand strict security guarantees, such as national elections. This prototype is also explicitly not intended to prevent certain large classes of attacks involving denial of service, detectable corruption of the election results, vote selling or voter coercion.

Compared to other verifiable election providers currently available (such as Helios Voting[2]), PloneVote seeks to provide integration with information systems based on the Plone CMS platform. We believe that such integration will increase voter usability and ease of configuring an election in scenarios in which voters are already registered as users on the Plone system used by the party hosting the election. This is the case of IMATE, which is our initial target for deployment of the system.

2.2 Definitions

2.2.1 User Roles

The system contemplates six different types of users or user roles. Each user role is a constrain on how an user can or is expected to interact with the PloneVote system. With the exception of the *Server Administrator* or *System Administrator* role, roles exist with respect to a particular election being run in the system, rather than the system as a whole. Many different persons may have the same user role. A single person might, depending on election and institutional policy, have multiple roles in a given election.

- *Voter*: A voter is an user authorized to cast a vote in the election. He has an account and log-in credentials for the Plone site in which the election is hosted. Voters are not assumed to have in depth knowledge of the election process or training in using any specialized election programs other than that necessary to interact with the Plone CMS.

- *Candidate*: In elections in which the result of the voting process is the selection of a person or group of persons, a candidate is a person who can

be selected so. Candidates are expected to have an account and log-in credentials for the Plone site in which the election is hosted, though the system might optionally support candidates that do not have an account on the system. By virtue of being a candidate, an user should not get any further privileges in her interaction with the system, other than the ability to accept or reject her candidacy.

- *Election Official* or *Election Trustee*: An Election Official or Election Trustee is one who holds part of the information required to access or decrypt the election ballots. The idea is that votes can only be counted when a predetermined proportion of the Election Officials or Election Trustees collaborate to do so. Election Trustees as a whole are trusted to only perform decryption and vote counting over an anonymised collection of ballots, which cannot be associated to the voters who casted them. The collective of users with this role is also referred as the Election Commission.

- *Election Administrator*: The Election Administrator is an user who configures or set-ups the election parameters and who might be able to trigger some phase transitions for the election during its run, depending on initial configuration. The Election Administrator might or might not be an Election Trustee as well.

- *Server Administrator* or *System Administrator*: A Server Administrator or System Administrator is any user who has privileged access to the underlying platform hosting PloneVote, including: Plone, Zope, the Zope DB, the file system or operating system of the server in which the election is hosted. Also anyone who can, legitimately or not, read private objects associated with the election or the PloneVote running product, as well as modify or inject code into the Plone instance hosting the election. This definition should be taken to include the programmers of the PloneVote system and any party who has physical or privileged logical access to the server running PloneVote server-side components.

- *Auditor*: An auditor is an user interested in determining the security of the running PloneVote system or the integrity of a particular election. Auditors should be able to access all data needed to verify the integrity of the vote counting process[1] and capable of performing actions that make it harder to subvert the election process. For optimal security, all persons should be potential auditors, including those who possess no other role in the election (not even that of voter). This demands that the system cannot be adversely affected by any number of malicious auditors.

[1] In all researched voting schemes, only voters can verify their own vote casting while maintaining privacy.

2.2.2 Code domains

Executable code in the PloneVote system runs in two different environments: the election server and client machines.

- *Server code*: Server code is that which runs on the server or servers hosting the election's data and Plone CMS instance supporting the election process. This includes all Plone products which are part of the PloneVote system and any other products or libraries used by those. Server code is ultimately under the control of the Server Administrator. This code can be leveraged to prevent certain user actions on the system without resorting to cryptographic methods or distributed validation. On the other hand, server code can be subverted by those with the Server Administrator role, who have privileged access to the system, and cannot be fully trusted by voters.

- *Client code*: Client code runs on a machine controlled by the party executing an action, such as the personal computer of an user. Each user can, in theory, ensure trust of the client code they run. On the other hand, no other user can verify the actions of client code run by another user, and a malicious user can modify the actions of any client code he personally executes. As such, we say that client code acts on behalf of the user running it, and can be trusted only as much as said user is. Client code includes: desktop applications, command-line tools and JavaScript code downloaded by the web components of PloneVote, so long as the code is ultimately executed by a machine controlled by the user who is served by that code.

2.2.3 Election primitives

In addition to users and code domains, there are some fundamental notions involved with the PloneVote system at the conceptual level. Some of the most important are:

- *Election Process*: The full protocol, actions and constrains followed in realizing an election from beginning to end using PloneVote.

- *Election Security Protocol*: The election security protocol describes the basic rules of interaction between users designed to provide a certain set of security guarantees on the Election Process. For purposes of the current version of PloneVote, those guarantees include only voter verifiability of the election results and secrecy of vote. This core protocol uses cryptographic techniques and similar mechanisms to ensure that the security guarantees of the election are satisfied. The election security protocol does not necessarily include all the details of the Election Process, just those that directly involve PloneVote's election security guarantees.

- *Ballot*: The representation of a cast vote within the system.

- *Ballot Scheme*: The class of vote representations accepted in a particular election. The Ballot Scheme constrains the possible Ballots users might cast as their vote. For any particular election, the Ballot Scheme can - and often will - be additionally constrained by parameters, such as the number and names of the candidates.

- *Vote Counting Scheme*: The process by which cast votes are added together to determine the election results. Both the Ballot Scheme and the Vote Counting Scheme must be set before any vote is cast. In many cases, a Ballot Scheme implies its Vote Counting Scheme.

- *Election Phase*: A particular stage of the Election Process in which only a specific subset of actions is allowed by the system. The number and constrains of Election Phases may be influenced by the Election's Configuration. Transitions between Phases might be time triggered, action triggered or require manual execution by the Election Administrator.

- *Election Configuration*: All properties of an Election which can differ from those of other Elections, such as: the Election Phases, together with their constrains and transitions; the Ballot Scheme and Vote Counting Scheme; the Plone users associated with each User Role, etc. Election Configuration is also the first Election Phase for most elections and realized by the Election Administrator.

- *Running Election or (just) Election*: A particular Election Process with a certain Election Configuration and associated data, which is being executed within the PloneVote system. The system can host and run simultaneously multiple Elections, which should not be capable of interfering with one another[2].

3 System concerns

3.1 Orthogonality of concerns

Broadly speaking, the requirements of the Election Process that PloneVote must support can be classified into three separate areas of concern:

- Election security: What security guarantees does the system give? How are they enforced? What steps must be followed as part of the Election Security Protocol to force those guarantees to hold or at least verify whether they hold or not?

- Voting scheme: How do users encode their preferences into a vote (or multiple votes, where allowed)? How do votes get recorded? How are votes finally counted and the result determined?

[2]Except possibly by competing for resources. Denial Of Service attacks are explicitly not considered in our current threat model.

- Election running: How does one set up an election or configure its options? What are the Election Phases? What administrative guidelines or rules must the election follow (voter and candidate selection, span of time to vote, etc)?

In some cases, this concerns overlap. For example, the Election Security Protocol might require a few mandatory election phases and phase ordering in order to make any reasonable guarantees of privacy (e.g. a ballot mixing phase before the decryption phase). However, for the most part, this concerns can be made to be independent - or orthogonal - from one another. The components of the system dealing with security need not know how votes are described into ballots or counted afterwards, so long as they can verifiably record and anonymise a collection of arbitrary text ballots. Also, policy defining administrative requirements for the election needs not be understood by the security components, so long as certain mandatory phases are included; timed constrains have no relevance to the Election Security Protocol. Finally, defining how ballots are described and counted can be made so that it requires no knowledge of the election's security, administrative or user interaction protocols.

For purposes of the current version of PloneVote, we will seek to deliver a system with a single well-understood Election Security Protocol, which - although coded in an extensible way, allowing for future improvements - is not designed to be fully replaced or changed from election to election. The Election Security Protocol must thus be minimal while achieving the security guarantees we require for all elections.

On the other hand, PloneVote is to be designed so that diverse voting schemes (both for ballot casting and counting), can be easily added into the system and be used for different (possibly simultaneous) elections. For the current prototype, however, only one particular voting scheme will be supported, for the sake of simplicity.

Finally, election running and configuration should be flexible enough to support the administrative requirements of all elections commonly effected within the IMATE and many more elsewhere, without requiring new code to be added to the system. Modifying an election's phases and administrative behavior, should be accomplished simply by selecting the appropriate options at election set-up time.

To support the desired extensibility in voting schemes, election running phases and administrative requirements, we recommend that the system be designed in such a way that components dealing with each of the above three concerns are very loosely coupled to those in charge of another concern, and interact only through well-defined and general interfaces. In the next few sections, we will detail the requirements for the system separated by concern.

Certain aspects, such as performance or usability, interact and intersect each of these concerns. But those aspects are less particular to the PloneVote system and can be solved by applying well-known patterns, good design practices and good coding practices.

3.2 Election security

3.2.1 Online elections security overview

Electronic voting is hard to get right, and online voting is even harder. There are two main reasons for the difficulty in designing an online voting scheme that is at once: well understood, easy to use and secure. The first is the great number of different, often conflicting, security requirements that may apply to an online voting system. The second is the lack of a trusted party or trusted computing base which can be relied upon to ensure any security guarantees.

For the first point, consider the following list of desirable security properties of an election system. These are properties that may be reasonable to expect from a truly reliable voting system used for national or state elections, or in some similar environment in which many parties can be expected to have an interest in subverting the election process:

Security property	Description
Democratic	The set of eligible voters is known at the beginning of the election process, either by enumeration or by description. Only eligible voters can vote, and no eligible voter can be prevented from voting. The number of votes each voter can cast is also known at the beginning of the election and no voter can cast additional votes (that are counted) beyond that known allowance.
Private	No one can tell how a voter actually voted, except the voter herself.
Uncoercible	A voter cannot be coerced/bribed to vote a particular way. This implies that no voter can prove how they voted to another party.
Accurate	Submitted votes cannot be altered, no spurious ballots can be added and no valid ballots can be removed from count or miscounted.
Verifiable	Each voter must be able to verify that their vote was cast correctly. Anybody should be able to verify that all cast votes were counted correctly.
Robust	The election may not be disturbed by a small group, such as by a Denial of Service (DOS) attack, corruption of cast ballots or misuse of complaint procedures.
Fair	No partial results are known before the election is closed.

No known scheme with reasonable usability exists that perfectly satisfies all of the above security properties. In fact, goals like verifiability and uncoercibility (or even verifiability and privacy) seem to be in conflict. Even paper ballots provide at most partial protection for most of those properties, based upon bureaucratic procedure. In particular, traditional paper-based voting is not

strongly accurate, robust or fair, and is usually not verifiable at all by a regular voter.

Besides the complexity of the security goals, there is the second challenge of lacking any single entity who may be trusted to enforce them. For the voting process to be trusted by the voters, they must be able to know that security properties are maintained regardless of the actions of Election Officers, the Election Administrator or the System Administrators. It is also clear that no single voter can be trusted not to be an attacker, wishing to subvert the election in favor of a particular desired result. Finally, machines and software systems can only be trusted as far as the people who construct, operate or merely have unsupervised privileged access to them. Thus, no single principal involved with the election process can be trusted not to be malicious[3].

This all makes it clear that PloneVote cannot promise to deliver all seven security properties outlined in the table above, much less without some caveats regarding the proportion of users - with certain roles - expected to be honest. In section 3.2.2, we outline the guarantees we hope PloneVote's Election Security Protocol will deliver for all elections; in section 3.2.3, we spell out the constrains and limitations of those guarantees as well as the security properties we explicitly are not planning to support.

3.2.2 Desired guarantees

PloneVote's Election Security Protocol is geared towards offering two core security properties: verifiability and privacy. Those properties by themselves and the particular method of achieving them also have some interesting implications providing conditional accuracy, democracy and fairness.

Fully voter verifiable elections

Voters should be able to verify that their own vote was cast and recorded correctly, and that the final count is correct for all correctly cast votes. If all voters verify their own votes and at least one honest voter verifies the final count, this immediately implies accuracy for any complete election that passes verification without showing errors. An honest voter verifying their own vote and the final count should never get a passing result from a corrupted election, that is: one in which the count was made incorrectly, votes were added or subtracted or their own vote was altered.

This property is expected to hold in spite of any number of malicious users of any role (including Election Officials, the Election Administrator and the System Administrator) or any alterations of server code. The voter still needs to trust or verify all client code used by her personally, in order to ensure that the verification process is being done correctly. They might use client code

[3]For some properties and in some scenarios, though, it might be reasonable to trust that certain groups of people are not malicious as a group. That is, malicious and collaborating. For example, the Election Commission as a whole might be trusted further than any single election official by themselves.

produced by any trusted party or rely in auditors to the default tool's published source code to increase confidence on that code.

It may also be of importance to note that this property does rely on some unproven, but widely held to be true, cryptographic assumptions, such as the hardness of certain problems involving discrete logarithms. Since these assumptions are relied upon for systems as vital as online banking and defense infrastructure, we submit that we may consider them true at this point for the purpose of reasoning about the security of PloneVote for institutional elections.

Privacy

Unlike verifiability, privacy in PloneVote is not enforced unconditionally. However, in the absence of coercion, a voter's cast vote is guaranteed to be private so long as either of two conditions hold:

- PloneVote server components and any System Administrators (which includes anyone with privileged access to the Plone system, legitimate or not) are not malicious.
- A particular proportion (often the majority) of the Election Trustees are honest or - at least - not collaborating to betray voter's privacy. The particular number of trustees required to breach privacy is the same as those required to perform decryption of the cast ballots, which is a parameter of the Election Configuration.

Again, both conditions must be untrue before voting privacy can be breached and the association between cast vote and its caster can be made. Just a corrupt server or a corrupt Election Commission are not enough to subvert privacy, but both entities in collusion may.

Accuracy

As indicated above, PloneVote elections can be trusted to be accurate if and only if all voters verify their own vote, at least one honest party verifies the vote count, and all verifications are successful.

Democracy

A public list is to be kept by the PloneVote server of who is allowed to vote and who has voted in the election (without association to the particular vote, of course). This lists can be used to raise a complain when an authorized user is denied the right to vote, and he can always verify that his vote was cast and counted correctly.

Additionally, the second list can be used to verify the absence of added votes in impersonation of voter who might not have voted. This is part of the verifiability property, so long as authorized voters who did not vote also verify the fact that the list records them as not having voted[4].

[4] A voter may also nullify his vote in most ballot schemes, which results on him being

Fairness

While accuracy and democracy are dependant on verifiability. Fairness in PloneVote depends upon privacy. A trusted server guarantees fairness simply by refusing to release any partial results of the election until the phase in which votes are accepted has concluded. Furthermore, since a large proportion of the Election Trustees are needed before stored votes can be decrypted, even a malicious server cannot violate fairness by publishing the contents of a partially recorded election, unless the Election Commission is colluded in the act, and thus privacy can also be violated.

A malicious server, acting by itself, can still release some information, such as the order in which votes were cast or who has already cast a vote and who has not. This might conceivably affect fairness in some way less definitive than releasing a partial count would. On the other hand, this partial disclosure on the part of the server would be useless in attacking privacy (when the Election Commission is not colluded), as ballots are shuffled before counting.

Finally, using PloneVote does not prevent unofficial, non-verifiable polls that might reveal potential partial results by asking voters to volunteer who they voted for. This may affect fairness, and is not preventable by any reasonable voting scheme.

3.2.3 Limitations of scope

First of all, it should be kept in mind that the current version of PloneVote is intended as a research prototype, attempting to provide a best-effort implementation of a secure elections system based on known protocols. As such, no security property, not even verifiability or privacy as described above, are to be considered truly guaranteed. However, there are also several security properties that are not provided within PloneVote by design, even assuming a bug-free implementation (which this prototype is unlikely to be, specially upon first release).

In normal usage, PloneVote provides no protection against coercion, although limited uncoercibility can be achieved if some additional requirements are imposed on how voters vote. Robustness is not particularly ensured or addressed by this prototype.

uncoercibility

The system by itself provides no guarantee of uncoercibility at the time of voting. That is, any malicious party that observes a voter casting her vote and obtaining the information required to verify it was cast, can then use that same information to verify that as the final vote of the voter, and can thus potentially coerce her into voting in some particular way. Allowing the user to overwrite her vote provides very little mitigation for this, as the system generates a receipt for

counted as having voted but a blank vote showing up during the counting process. This can be a way of extending full privacy to the decision of whether to vote or not.

each vote that can be used to prove that said vote was cast and counted, this receipt can be shown to the malicious party at voting time and then verified when the election results appear.

On the other hand, the receipt cannot be translated into the contents of the vote. It is only proof that the vote was cast correctly if the voter used trusted client code to cast said vote and recorded the receipt as given by that code. Then the receipt can only be used to check that a vote corresponding to that receipt was counted, but can't be associated with any one decrypted vote from the counting process (except by collaboration of the same proportion of Election Trustees described as threatening privacy, although this time without server involvement). As such, the vote receipt cannot be used to prove to any third party that a vote for certain candidate or preference was cast, unless said third party also observed the vote casting process. Because of this, some protection against coercion can be obtained if voting is only allowed from a physically secure location in which the voter is both guaranteed and forced to cast her vote unsupervised. In that case, coercion is prevented unless the majority of the Election Commission is corrupt.

In the normal usage case for PloneVote, however, uncoercibility is not addressed.

Robustness

Robustness in PloneVote is equivalent to that of most web applications: unprivileged users (all but Server Administrators or a significant number of Election Trustees) should not be able to stop an election with their actions when following or appearing to follow protocol; a malicious or compromised server can, on its own, prevent the election from completing successful; and Denial of Service attacks against the server or network infrastructure might prevent voter from voting, stopping the election as well. Besides, a minority of the Election Commission may jointly refuse to decrypt or count the votes. As such, PloneVote is not considered to provide any security guarantees regarding robustness, although common web server securing techniques and careful selection of trustees can provide some resilience in that regard.

The Election Security Protocol is designed in such a way that the server can verify that no user corrupts any ballots already recorded. Provided an honest server, users cannot garble other users votes, although they might cast a garbled vote themselves, which would then go uncounted[5]. A malicious or compromised server can corrupt the election results at any point before the final count is made and may also deny verification[6], invalidating the election. As such, a Server Administrator can clearly obstruct an election.

Since a pre-determined proportion of Election Trustees is required to decrypt

[5] Provided client code should forbid this, so a careless user is unlikely to submit a malformed vote, but a voter determined to do so may.

[6] It cannot, however, fake the results of a verification in such a way that it appears to be valid but is not. Verification is realized in client code with only some evidence of correctness being provided by the server.

the ballots before counting, any number of trustees larger than the reciprocal of those needed for decryption, can prevent vote counting simply by withholding their private keys.

It is also likely that for the initial implementation of the system and using a Plone server with a standard set-up, any determined attacker can obstruct the election as well, by using a Distributed Denial of Service attack. Isolating the server inside an internal network and allowing voters to vote only from within that network can be a potential course of action as soon as a DDOS is detected. A feature for future versions of PloneVote that could greatly increase robustness, would be the ability to run the server side system in a distributed manner on multiple servers simultaneously and have all servers record all ballots and evidence required to count and verify the election.

3.2.4 Election Security Protocol

The Election Security Protocol for PloneVote is based mainly on Josh Benaloh's Simple Verifiable Elections[1] and the Helios Voting System[2]. The following sections describe the steps taken by the security protocol in brief, separated in four stages: before voting, while voting, after voting and verification after the results have been published. Certain complex cryptographic operations are assumed to be possible, such as verifiable shuffling or mixing and threshold encryption. This operations are described in [1, 2, 3].

Previous to the voting phase

1. The Election Commission is selected, together with a fixed number k of the n members that will be needed to decrypt and count the votes in the end.

2. Each election officer uses client code to generate their private key (kept secret on their personal machines) and commit a partial public key for threshold-encryption to the server[7].

3. Some party (possibly the server itself) generates a full public key for the threshold-encryption scheme, using the partial public keys of all election officers. The scheme is set-up so that any data encrypted with that key can only be decrypted using k distinct private keys from members of the Election Commission. That the public key is correct for this scheme can be tested by anyone, requiring only the partial public keys, which are considered publicly available information.

4. Election Officials should check that their public keys, as they appear in the server without log-in, are correct. Each official is also strongly encouraged to verify the final public key. Client code will be provided to automate this process.

[7] Conceivably, this pair of keys might be the same across multiple elections, provided the private key has never been compromised or shared with another party.

5. The list of voters is set-up, finalized and published on the server by the Election Administrator in accordance with the administrative process for the election.

6. All allowed voters are marked on the server as able to vote and having not voted for the election in question. Until the voting phase has ended, only the voter himself and the election administrator can see if said voter has or has not voted.

Ballots are generated by the server as well during this stage.

During the voting phase

For each voter:

1. The voter creates their vote using client code[8] and the ballot specified for the election (more on this when we review the voting scheme).

2. The voter uses client code to encrypt their vote with the election's public key.

3. A verifiable receipt is obtained as a hash of the encrypted vote.

4. The user submits the encrypted vote to the server (but not any part of the unencrypted vote).

5. The server re-calculates the receipt and stores the vote, tagging it with its receipt.

6. The server marks the voter as having already voted and allows no further votes from that user in that election (multiple votes per user can be implemented at the ballot description level).

7. The server erases all information which might connect the user to their vote, other than timing based channels. (For example, logs might show that the user has voted and when, but should not include the receipt or any information about the encrypted vote)

After the voting phase

1. Recorded votes are shuffled on the server, in a non-cryptographic way (just a re-ordering without re-encryption). The goal is only to prevent timing based channels from tying voters to their votes as they appear in the first ballot set, before verifiable shuffling.

2. The server publishes a list of all vote receipts and a separate list of all users who voted. No association is made there between user and receipt.

[8] Possibly JavaScript client code delivered from the server for usability reasons, but it still should run on the client system.

3. A period is given in which any interested party might verifiably shuffle[9] the current set of votes for the election using client code. The shuffled set is uploaded to the server together with the proof of correct shuffling. The server verifies the proof and substitutes this new set as the current set of votes for that election. All shuffling proofs, and intermediate encrypted collections of votes are made public by the server.

4. The server may demand a minimum number of shuffles by different parties, possibly requiring also a certain number of Election Officials to shuffle the votes. After the votes can be shuffled enough times, the Commission cooperatively decrypts only the most recently shuffled collection of votes and publishes the whole decrypted votes and a proof of decryption, together with the results.

Verification

1. Each user authorized to vote should verify that:

   (a) If they voted, the receipt for their vote is on a list of published vote receipts.

   (b) If they did not vote, they are not in the list of users who voted.

2. Any person may then verify the whole election by:

   (a) Verifying the first collection of votes: ensuring that the number of votes is consistent with the number of people recorded as having voted (and that those people are authorized to vote), as well as that the encrypted votes match their published vote receipts.

   (b) Verifying all proofs of shuffling from the first collection of votes to the final one (the one which got decrypted).

   (c) Verifying the decryption of the final collection and re-counting the votes.

Client code should be provided to ease full election verification.

3.3 Voting scheme

3.3.1 Rationale

There are two types of voting schemes used for elections within the Institute of Mathematics at UNAM, one is a form of equally weighted multivote, while the other is a preferential voting scheme. Internal regulations, which are difficult

[9] A verifiable shuffle is a process by which a collection B can be constructed from collection A, so that it can be proven that the collections are equal (in elements and number of repetitions of each element) without revealing the correspondence between elements in A and elements in B.

to change, prescribe which scheme is used for which elections, with the most common being the multivote model.

As stated before, PloneVote seeks to make the security and process of the election independent from ballot representation and vote counting, as well as to implement multiple schemes for both, which can be used simultaneously for different elections on the same system. However, for our first prototype, we also aim to reduce the amount of work required to implement parts that are not fundamental to the security, extensibility or usability of the system. As such, we have decided to require implementation only of the multivote scheme, with the option of adding preferential voting after the first development iteration. This helps the development process, since the multivote scheme is the simplest of the two, and also takes usage considerations into account, since this is the model to be used in the next scheduled elections at IMATE.

3.3.2 Ballot scheme

For our multivote voting extension, the ballot scheme takes a list of candidates and options, each with a name and a picture, and possibly some other information such as an url to a campaign platform page. The ballot scheme also takes two parameters: *min*, the minimum number of options a voter can elect to vote on, and *max*, the maximum number of options the voter may cast a vote for. The ballot is presented to the user as a grid of candidates (shown by name and picture, in alphabetical order), where she can check or uncheck candidates. The vote should only be considered valid if the number of checked candidates are between *min* and *max*, in which case the ballot will represent a single vote of equal value for each of the checked candidates.

The client code should attempt to prevent the user from selecting an inappropriate number of candidates, although, being in the user's control by design of the Election Security Protocol, it cannot forbid casting a ballot with an out-of-range number of candidates. Such ballots will be considered invalid and not counted at vote counting time, although they will be published with the full decrypted results of the election.

Another parameter to the ballot scheme is a boolean flag to indicate if a blank vote is permitted. If so, the user will have an additional option to vote blank at vote time, which will create and cast an empty vote with no value and no candidates listed. Note that allowing a blank vote allows a voter to cast a vote for less than *min* candidates, but only if that vote is for exactly 0 candidates. This is intended as a mechanism to ensure privacy of the decision not to vote, since the system publishes a list of voters who have voted. A voter emitting a blank vote will be treated by the system as having voted and obtain a voting receipt as she would if she had cast a meaningful vote.

Write-ins are not permitted in this ballot scheme. This is just a design choice, not an inherent limitation of PloneVote. In theory, an option to add write-ins to multivote could be easily implemented.

3.3.3 Counting algorithm

The counting algorithm for multivote is trivial:

1. First, invalid ballots are detected, counted and discarded from the counting process[10].
2. Second, blank ballots are similarly counted.
3. For each of the remaining ballots, the candidates checked in the ballot receive each a single vote.
4. The results are published as a the total number of votes counted for each candidate, the total number of blank ballots counted, and the total number of ballots detected as invalid. Candidates are ranked by the number of votes they received and winners are to be selected as the first candidates in rank order up to a certain cut (e.g. if selecting three members for a committee, the three highest-ranked candidates should be chosen). Selecting the winning candidates is an act external to the system, as the results only show the candidates in rank order.

The number of invalid ballots and blank votes are published for completeness, but they have no bearing on the election's decision, other than perhaps indicating potential problems with the voting process (if an statistically significant number of invalid ballots are detected) or electorate dissatisfaction with all candidates (if many blank votes are detected).

3.3.4 Optional: Preferential voting

An interesting voting scheme that could also be implemented, and which may eventually be needed by the IMATE, is preferential voting. It should operate with the same parameters as multivote, but allow the voter to give an ordered list of between min and max selected candidates, instead of checking an unordered set. After capturing voter preferences so, there are a number of different counting methods to determine the outcome of preferential voting. Currently, Schulze-STV is suggested as an interesting option.

Preferential voting with Schulze-STV counting is significantly more difficult to implement than multivote, and could be added to an extensible system as a module or collection of modules. It is marked as an optional feature for the current prototype, but a working PloneVote system that can later be extended to support different voting schemes is more important at the moment.

3.4 Election running

3.4.1 The standard candidate election

The aim of PloneVote is to decouple the mechanisms related to securing arbitrary elections from those related to running any particular type of election. However,

[10] They should, however, still be published with the election results.

for our first prototype release, we will support a single particular model of election. We will refer to this model as the Standard Candidate Election or SCE, which is described in this section.

Each model of election is a general schema of how an election must run from start to finish, and which actions should be performed by each of the parties involved in the election. The SCE is a very general model, supporting most elections in which a certain group or organization wishes to elect either a single of its members or a subgroup of them (a committee, for example). It is the explicit goal of the SCE model to support any election of candidates currently required by the IMATE, and as many other scenarios as possible.

The SCE encodes certain administrative requirements as part of the process that will be hard-coded into the Plone product used for running this type of election (e.g. steps like key set-up or the use of candidate and voter lists). However, most of the requirements of a particular election are implemented as configuration parameters to the SCE election in question, which are given in the first step, when creating the election. For example: the criterion used to authorize voters and candidates, the timing of each phase and deadlines for each step, the type of ballot and counting algorithm to use, and many more, are configurable options.

A few things are intrinsic to SCE in its current form, that may not hold for every possible election involving choosing a group of people. One such inherent property, is that votes are assumed to be always equally weighted; that is, no voter may vote more than once and the votes of every voter carry exactly the same weight. A different model would be required for elections in which voters are explicitly and legally considered to have different power to influence the election.

It is possible to use SCE to vote on something other than candidates that are Plone users, simply by manually creating the options for all "candidates", which can stand in for: people not in the system, courses of action, plans, projects, laws, etc. The main usage case is voting for people, however.

3.4.2 Phases of a SCE

There are four main phases of an SCE. Whether changes from one phase to another are time triggered or manually triggered by the administrator, as well as the values for the timers, should be configured at election creation.

1. *Set-up Phase*: In this phase, the election is created on the server and its basic configuration is given by the election administrator. The election commission is selected and the threshold-encryption scheme is initialized. A list of candidates and a list of voters are created, published and given enough time for complains or corrections to be addressed. Finally, the lists are finalized and published as non-modifiable, the ballot template is created and the internal state of the server is set-up to commence the voting phase.

2. *Voting Phase*: During this phase, voters are allowed to submit their votes.

Recorded votes are withheld by the server as an additional security measure, until the counting phase. Before passing from this phase to the counting phase, votes are non-cryptographically shuffled in the server to prevent timing attacks.

3. *Counting Phase*: A period of time is given in which any user may cryptographically shuffle the votes recorded, and minimum number of shuffles is enforced. After that, election officials may submit partial decryptions to the most recently shuffled set of votes, the server will then combine the partial decryption into a full decryption of the votes and perform the count. All proofs required to verify the election are published as they are created during this phase. In the end, a list of users that have voted and a list of vote receipts are published.

4. *Verification Phase*: For a predetermined time after vote counting has finished (and before the results of the election are considered final), the proofs necessary to validate the election, as well as all the shuffled vote sets, must remain public in the server. Voters and external auditors are encouraged to verify the results of the election themselves, using provided or third-party client code.

3.4.3 Steps of a SCE

There are many steps in realizing an SCE, which are performed by multiple different parties. Some steps may be done simultaneously, while others depend on a certain step having taken place before itself. For some steps, the system should allow to configure a span of time between the time in which the actions required to fulfill that step become available, and the time the step is considered to have been taken regardless of user action (e.g. for complains about the list of candidates or for voting).

Here we show a diagram of the dependencies between steps and the phases in which they take place (Figure 1), as well as a description of each step:

1. Election Creation: The election administrator logs into the server and creates an election. Anyone with the appropriate Plone permissions can create an election object, and becomes automatically its election administrator. During creation, multiple parameters of the election are configured, including: name, description, type of election (e.g. SCE), type of ballot and counting algorithm, timers for steps and phases and number of election officials/trustees (both in total and those required to decrypt the votes).

2. Election Commission Selection: The election administrator selects the Plone users who will act as election officials, in accordance with the numbers given at election creation time. Members of the election commission may only be modified before key set-up has been finalized.

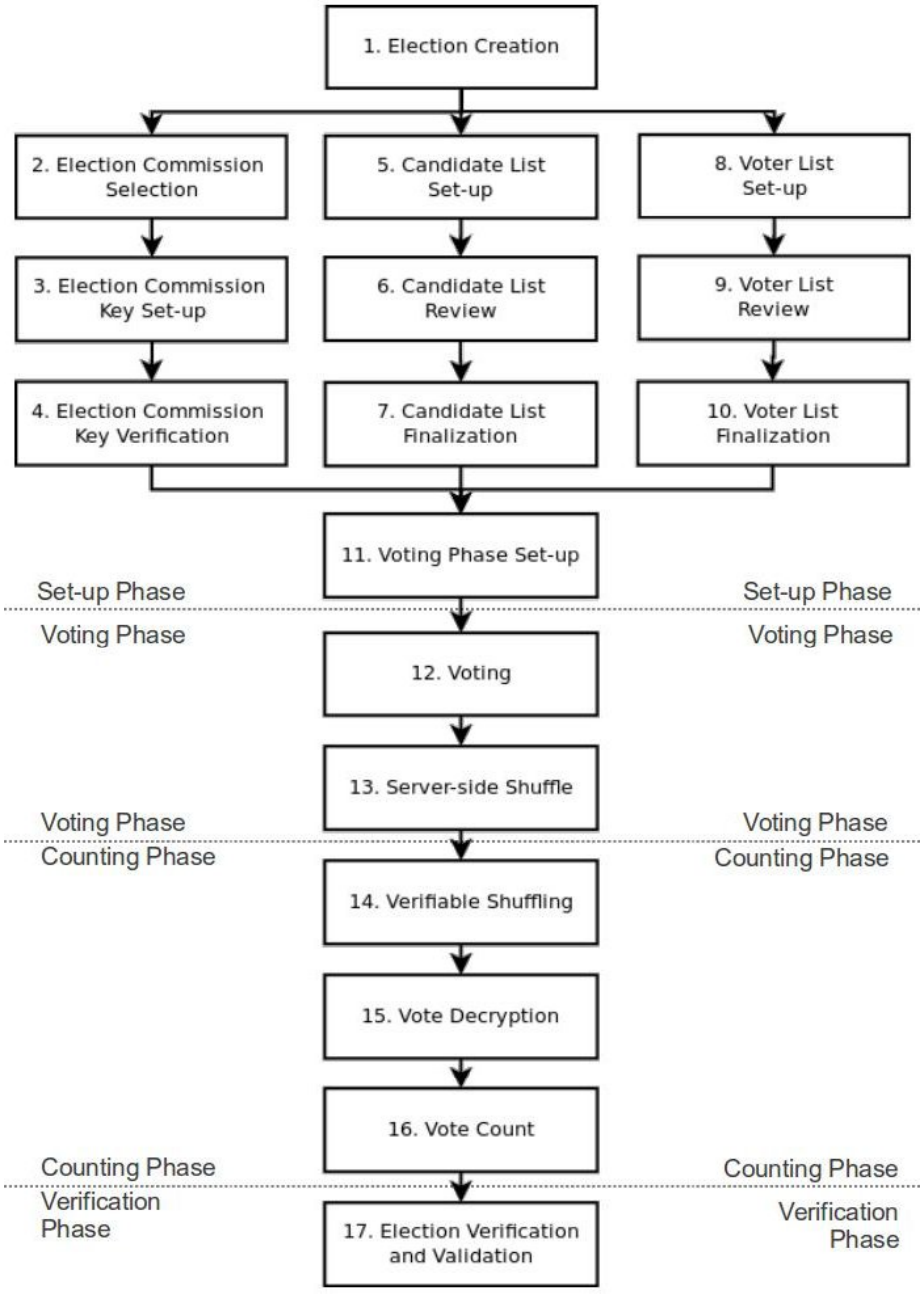

Figure 1: SCE steps and phases

3. Election Commission Key Set-up: Election officials use client code to produce their own private keys and partial public keys for the threshold-encryption scheme. They submit their partial public keys to the server. A combined public key for the election is configured as described in 3.2.4. From this point onwards, private keys must be stored safely by each election official.

4. Election Commission Key Verification: A period of time is given for election commission officials to verify that their partial public key was correctly recorded by the server and that the combined public key was generated honestly. Client code for doing so will be provided.

5. Candidate List Set-up: The election administrator either manually populates a list of candidates, or configures a query of properties that creates a list of candidates from the set of Plone users. Both kinds of lists can then be edited to add or remove candidates.

6. Candidate List Review: The candidate list is published for review, all users included in it are alerted. A period is given for clarifications and complains.

7. Candidate List Finalization: The candidate list becomes read-only for the election.

8. Voter List Set-up: The election administrator configures a query of properties that creates a list of authorized voters from the set of Plone users. This list can then be edited to add or remove voters.

9. Voter List Review: The voter list is published for review, all users included in it are alerted. A period is given for clarifications and complains.

10. Voter List Finalization: The voter list becomes read-only for the election.

11. Voting Phase Set-up: All voters are marked as having not yet voted and internal structures of the server are set-up so that the voting phase can commence. All previous steps are checked for error, and the lists of voters and candidates are verified to be public and read-only.

12. Voting: Voters may submit their votes as described in 3.2.4. After a voter votes, she is marked as having done so and not authorized to vote again.

13. Server-side Shuffle: The server non-cryptographically shuffles all received votes and publishes them in encrypted form.

14. Verifiable Shuffling: A period is provided in which any user can shuffle the votes cryptographically using client code (3.2.4). A minimum amount of shuffle operations may be required before vote decryption is accepted by the server.

15. Vote Decryption: Election trustees retrieve the last shuffled set of votes and create a partial decryption using client code and their private key. Partial decryptions are send to the server and combined there. Full vote decryption is performed.

16. Vote Count: The votes are counted according to the selected vote counting algorithm.

17. Election Verification and Validation: See verification phase in 3.4.2 and security protocol at 3.2.4. The list of vote receipts and the list of people who voted are published at the start of this step or the end of the previous one.

3.4.4 Logging and auditing

Besides publishing all information necessary for cryptographic verification of the election, the server should keep one or more logs of events that make it easier to diagnosticate the cause of invalidity of an election, or some problems of administrative nature (e.g. an user is incorrectly marked as having voted while she has not, or someone has subverted Plone security to vote on someone else's behalf, etc).

Again, the server is not fully trusted for ensuring the absence of irregularities or miscount in the voting process. Election verification and validation rely on cryptographic principles and the Election Security Protocol, in which all voters verify their voter receipts and at least one honest auditor verifies the count. However, logging certain events may facilitate diagnostics in case of irregularities and provides some added transparency to the process and information for the election administrator.

The system must log at least the following details, preferably to an append-only log file which Plone cannot delete:

- Start and end of each phase and step and whether it was an automatic, manual or time-triggered transition.
- The creation of a new election and the most important details of its configuration (name, type of election, ballot and counting algorithm, election trustees configuration, etc).
- The addition or removal of an election officer
- The submission of a partial public key for each election officer and a fingerprint of the key
- The time of creation and a fingerprint of the election's public key
- The finalization of the lists of voters and candidates, including: criterion for automatic lists, manually added and excepted entries, a fingerprints (or the full contents) of both lists.

- Date, time and user name for each submitted vote. This entry must NOT include any id of the vote or its receipt for privacy reasons.
- Fingerprints of all created encrypted sets, time and user of all shuffle operations and either proofs of correct shuffling or a fingerprint of the proof text.
- Time and user for each submitted partial decryption.
- Proof of decryption or its fingerprint.
- Published election results.

All log entries must include a timestamp and an unique id of the election they correspond to.

4 Use cases

4.1 Voter use cases

Following are described the common interactions of a voter with the PloneVote system, in the form of use cases. All this cases are for the first version of the PloneVote system, using the Election Security Protocol described in 3.2.4, the ballot and vote counting schemes described in section 3.3 and the SCE election model of section 3.4. Modules adding support for other types of elections, ballots and counting schemes may add new use cases or significantly alter the described ones, but those extensions are beyond the scope of the first PloneVote prototype.

Voters are also encouraged to take the role of auditors for the election if they have the time and feel comfortable with using the auditor client code. The more users acting as auditors, the harder it is to subvert the trust of the election. A single honest and capable auditor ensures an accurate election.

4.1.1 Verifying voter status: status initially granted

In this use case, an election X has been created by user Alice. User Bob is selected, either automatically or manually, into the initial voter list for the election. The following scenario takes place during step 9 of the SCE, in the Set-up Phase.

1. Alice has created the voter list for election X and published it for review, Bob appears in that list.
2. The system notifies Bob via a message in the Plone interface whenever he is logged into the Plone site. Additionally, the system may send Bob an e-mail or some other notification. In either case, the message contains the name and description of the election and a line letting Bob know that he is authorized to vote in that election. The message also contains the time left before the voter list finalization step of the election, if such election step

is time-triggered for election X. After voter list finalization takes place, actions for this use case are no longer available and this scenario goes to step 8.

3. A link to the full voter list for review is also provided in the notification message. Bob may privately raise complains about other voters in the list, but should do so by a channel outside the PloneVote system (e.g. e-mail).

4. If Bob feels that he should not be an authorized voter for that election, and the list is in error, he goes to the next step (5). Otherwise, he optionally marks the notification as received (which makes it disappear from the interface and possibly logs acceptance) and the scenario goes to step 7.

5. Bob notifies the election administrator that he should not be considered as a voter for that election. That may be done through any channel (e-mail, for example). Optionally, the system could present an action to mark oneself for delisting from the voter list, allowing the user to add a brief message explaining why he shouldn't appear as a voter. Only authenticated users can mark themselves for delisting, an user cannot mark an user for delisting from the voter list other than themselves.

6. If Alice agrees with Bob's delisting, she may remove him from the voter list. If there's a mechanism in the interface to request delisting, that should cause a notification to be sent to the election administrator (Alice), together with a single-click option to honor the request.

7. If for some reason Bob is removed from the voter list (whether he requested it or not), he should immediately be sent an automatic notification of the fact. At this point, the system proceeds as it would if starting from scenario 4.1.2.

8. After the voter list finalization step of SCE takes place, the system sends one last notification to all users authorized to vote in Election X. This notification can be acknowledged as received (which makes it disappear from the interface and possibly logs acceptance). No other action needs to be taken.

4.1.2 Verifying voter status: status initially denied

In this use case, an election X has been created by user Alice. User Bob is **not** selected, either automatically or manually, into the initial voter list for the election. The following scenario takes place during step 9 of the SCE, in the Set-up Phase. After voter list finalization takes place (step 10 of SCE), actions for this use case are no longer available.

1. Alice has created the voter list for election X and published it for review, Bob does not appear in that list.

2. Since Bob is not in the voter list for election X, he receives no notification about that election, other than perhaps through the Plone site news or other users. He can, however, see the election's public page and the candidate voter list published for review.

3. Bob may privately raise complains about voters in the list other than himself being included or excluded, but should do so by a channel outside the PloneVote system (e.g. e-mail).

4. If Bob is content with not being a voter in that election, this use case has ended. Otherwise, continue to next step.

5. Before the voter list has been finalized, Bob may raise a complain, requesting inclusion into the voter list. This may be done outside the system (e.g. sending an e-mail to Alice asking her to add him). Optionally, there may also be a way of requesting inclusion from within the system, by using an option of the view for the voter list and providing a brief explanation of why Bob should be authorized to vote in election X.

6. If Alice agrees with Bob's request, she may add him to the voter list. If there's a mechanism in the interface to request addition to the list, that should cause a notification to be sent to the election administrator (Alice), together with a single-click option to honor the request.

7. If for some reason Bob is added to the voter list (whether he requested it or not), he should immediately be sent an automatic notification of the fact. At this point, the system proceeds as it would if starting from step 2 of scenario 4.1.1.

8. After the voter list finalization step of SCE takes place, if Bob is not on the voter list, he is not considered a voter for the rest of election X and thus receives no notifications about it, unless he is involved with the election in some other role.

4.1.3 Voting

User Bob appears in the voter list for election X. Election X is on the voting phase, during the voting step.

1. As the voting step begins, all voters are notified (site and/or e-mail) that the election's voting phase has been started and the exact time span within which voting will be permitted.

2. At any point during this period, Bob can access the election's voting page. For example, by following a link on his notification or by going directly to the election's public page and following a link to the voting page. To access the voting page a user needs not be logged into the Plone system, logging-in is requested just before submitting the vote.

3. The voting wizard for the selected ballot scheme is started. This is javascript code that generates the vote contents before encryption. It acts as if it were a series of views that must be traversed one after the other:

   (a) A first view asks Bob if he truly wants to start the voting page. This view should also check that the user's browser can handle the javacript client code required for voting[11].

   (b) Then, Bob is presented with the list of candidates and asked to chose and ordered list of between *min* and *max* preferred candidates. Where *min* and *max* are configurable parameters of the election.

   (c) Bob is asked to confirm his selection and shown a graphical representation of his vote (e.g. an ordered list of candidate pictures). Optionally, link to show the unencrypted vote in plain text or xml may also be available. In this view, Bob is also given the option to proceed to encrypt his vote.

4. The vote is encrypted, still in client code and without ever sending the unencrypted vote to the server. Bob is shown a verifiable receipt for the encrypted vote and asked to store it safely.

5. If not logged in, Bob is asked to log into Plone.

6. Bob is given one last option to confirm or cancel his vote. After confirming, the server checks that Bob is authorized to vote in the election; if so, it stores the vote and marks Bob's account as having already voted, updating internal lists accordingly and logging the event.

7. Bob is shown a page confirming that his vote was cast successfully.

4.1.4 Viewing election results and verifying one's own vote

Bob is a voter in election X, which has recently been decrypted an counted by the election commission, phase 17 of the SCE has just been reached.

1. The system notifies Bob that the election has been counted, via site messages and or an e-mail. The notification links to the public election's results page.

2. The election's results page contains the formated results of the election (e.g. the candidates in order, each with his or her vote/point count and those that have been elected marked as such) and a prominent link asking users to verify that their vote was cast correctly.

[11] Checking that JS is enabled is a minimum. Also, it's heavily recommended to test for standard DOM elements and javascript APIs, rather than perform agent string checks.

3. If an user follows the link to verify their vote, they are presented with a page including the full list vote receipts and the list of users who voted in the election (entries in one list are **not** associated with those in the other), together with instructions asking them to verify that:

   (a) If they voted, their voting receipt as they recorded it at vote cast time is on the vote receipt list and they appear in the list of users having voted.

   (b) If they did not vote, they do not appear in the list of users having voted.

4. A link is presented to leave the vote verification page back to the Plone system's main page. After Bob has verified that his own vote was cast correctly, he might either trust that an honest auditor will verify the count or use the auditor client code to do so himself.

Both the election results page and the verification page should be public, which means that Bob should not need to log-in or authenticate with the system. He also shouldn't need to submit his voting receipt to the PloneVote system, just manually verify it appears on the published list. This requirement makes it harder for the server to provide different lists to different users in attempt to hide spurious votes. The auditor client code enforces this precautions even further by never logging-in into the PloneVote system but merely downloading the complete election's verification information.

4.2 Candidate use cases

For the following use case, Charlie is a candidate for election X, while Alice is the election administrator.

4.2.1 Verifying and accepting candidacy

In this use case, an election X has been created by user Alice. User Charlie is selected, either automatically or manually, into the initial list of candidates for the election. The following scenario takes place during step 6 of the SCE, in the Set-up Phase.

1. Alice has created the candidate list for election X and published it for review, Charlie appears in that list.

2. The system notifies Charlie via a message in the Plone interface whenever he is logged into the Plone site. Additionally, the system may send Charlie an e-mail or some other notification. In either case, the message contains the name and description of the election and a line letting Charlie know that he has been selected as a candidate for the election. The message also contains the time left before the candidate list finalization step of the election, if such election step is time-triggered for election X.

3. If Charlie feels that he should not be a candidate for election X, and the list is in error, he goes to the next step (4). Otherwise, the scenario goes to step 6.

4. Charlie notifies the election administrator that he does not wish to be a candidate for that election. That may be done through any channel (e-mail, for example). Optionally, the system could present an action to mark oneself for delisting from the candidate list, allowing the user to add a brief message explaining why he shouldn't and does not wish to be a candidate. Only authenticated users can mark themselves for delisting, an user cannot mark a candidate for delisting from the candidate list other than themselves.

5. If Alice agrees with Charlie's delisting, she may remove him from the candidate list. If there's a mechanism in the interface to request delisting, that should cause a notification to be sent to the election administrator (Alice), together with a single-click option to honor the request. Delisting ends this scenario for Charlie, unless he is added to the candidate list again, in which case the scenario restarts itself from step 2.

6. Depending on election configuration, candidates may be required to confirm their candidature. If such is the case, the message received in 2 will warn Charlie that he needs to accept his candidature, for which a simple action will be provided (such as a link or button). After confirming his candidature, Charlie may not request to be delisted from the list of candidates for the election by using the system, though Alice, as the election administrator, still has the option of manually removing him. If confirmation is required for the election, and a candidate fails to confirm his candidature before the candidate list finalization step, he will not be considered a candidate for election X. (Acceptance of the candidature is publicly shown in the candidate list during step 6 of the SCE). If acceptance is not required for election X, Charlie may simply mark the notification as received (which makes it disappear from the interface) and consider himself a candidate for the election unless he gets a later message informing him he was removed from the candidate list.

7. If for some reason Charlie is removed from the candidate list at any time (whether he requested it or not), he should immediately be sent an automatic notification of the fact. If he thinks he should be a candidate, he may complain to Alice using a channel outside the PloneVote system and she may decide to add him again to the list, returning to step 2. Otherwise, Charlie stops being a candidate for election X at this point.

8. After the candidate list finalization step of SCE takes place, the system sends one last notification to all candidates in election X and publishes the final list of candidates. This notification can be acknowledged as received (which makes it disappear from the interface and possibly logs acceptance). No other action needs to be taken. No candidates or voting

options may be added, changed or removed after candidate list finalization takes place.

Candidate lists submitted for review should be public, allowing voters and other candidates to see the potential candidates and perhaps submit complains to the election administrator about the list. This will not be handled inside PloneVote for the current version, but the election administrator may add or remove candidates during step 6 of the SCE based on complaints received. It should also be clear to voters whether candidates need to accept their candidature and who has or hasn't done so. The final list should be clearly marked as such and probably sent in a notification to all voters as well.

4.3 Voting commission use cases

This are system use cases performed from the perspective of members of the election commission, either individually or as a group.

4.3.1 Verifying and accepting commission membership

In this use case, an election X has been created by user Alice. User Diana is chosen by Alice as an election trustee or election official. The following scenario takes place during step 2 of the SCE, in the Set-up Phase.

1. Alice registers the user names of those who are part of the election commission, including Diana. Alice must also specify the total amount of election trustees and the minimum number needed to decrypt the ballots.

2. Diana receives a message (site and/or mail) indicating that she has been appointed as an election official and asking her to confirm her acceptance of the position. A single-click action to accept is given (button or link) and another one to reject.

3. Alice receives a message indicating whether Diana has accepted acting as a trustee or not, which is also marked in some page belonging to the election configuration. If Diana rejected acting as an election official, Alice may name another user in her place.

4. If Diana accepts, she may proceed to the next scenario (key set-up and verification) and advance up to step 5 there on her own; step 7 requires that all commission members have accepted the role and uploaded their public key.

4.3.2 Key set-up and verification

This use case describes the step of negotiating the key set-up and threshold-encryption mechanism for election X, from the perspective of a designated election trustee (Diana).

This scenario assumes a desktop client that can log-in and interact with PloneVote as a web service client, which is our optimal case. For reasons of ease of implementation, it may be required, however, for the first prototype, that communication between the client code and the server system be accomplished instead by manually downloading and uploading files produced by one or the other.

1. Diana starts the election trustee client (a desktop application or similar client code). She points it to the PloneVote system by inputing the url of the site, then selects election X from a list of active elections.

2. Diana is presented a list of options possible for the election in its current state, amongst which there is the option for election official key-setup. She choses said option and authenticates to her account.

3. The election trustee client verifies she is an authorized election official for the election. Key set-up is denied to her if that is not the case and the scenario ends here.

4. The election trustee client generates Diana's private and public key pair and stores both on her current computer. Optionally, it could encrypt the private key with the same password as her Plone log-in. From now on, Diana can only act as an election official from that same machine, unless she copies the private key.

5. The election trustee client uploads and registers Diana's public key as a partial public key for the election. The server validates that she is an election trustee for election X before accepting her public key. Diana may then close the client and wait for other trustees to register their keys before step 7.

6. The server automatically generates the election's public key from the partial public keys, as soon as it has received one from each trustee.

7. At any point after the server has generated the public key for the election, Diana may connect with the election trustee client, select the same server and election, and chose the option to verify the election's key-set up. The client will then download all the partial public keys, ensure that Diana's public key is among them and is correct and redo the construction of the election's public key, comparing it to the one on the server. The client then gives Diana a report showing whether or not the server is using the correct public key.

It is required that all election trustees perform the first 5 steps of this scenario for any election during step 3 of the SCE. Also, it is heavily recommended that the last step be performed by all trustees previous to the end of the election's Set-up Phase.

4.3.3 Vote decryption and counting

After the voting phase has ended and the ballots have been verifiably shuffled enough times, the voting commission needs to decrypt the election and publish the results. Lets assume again election X with election administrator Alice and election trustee Diana.

1. The election must be marked as ready for decryption, forbidding any further verifiable shufflings. This may be a manual step realized by Alice or time triggered, depending upon the initial election configuration. Election configuration may also cause the server to forbid transition into this state if not enough shuffles have been performed.

2. Diana opens the election trustee client, gives the url to the PloneVote server, selects election X, chooses the option to decrypt the votes and finally is asked to authenticate herself. The client checks that she is an election trustee for the selected election. After authentication, the client may ask for confirmation before performing and submitting a partial decryption for election X.

3. The client checks that the election is marked as being in the ballot decryption step and downloads the latest ballot set, giving Diana the option to submit a partial decryption for the votes in there. The client may optionally perform other checks over the election, including verifying all shuffles and that there is the correct number of shuffles for the election, in a similar way as the auditor client would do[12].

4. If Diana choses to submit her partial decryption, said decryption is produced in her client machine using her private key. The partial decryption is then uploaded to the PloneVote server.

5. As soon as enough partial decryptions are available, the server will combine them to decrypt the votes and calculate the results of the election, publishing them together the voter and voting receipt lists. It will also publish all encrypted ballots, proofs of shuffling and the full decryption of the last set of ballots, thus allowing for the election to be verified and votes recounted independently.

6. The server will then send notifications (site and/or email) to all voters and candidates for the election, pointing them to the results page and asking them to verify at least their own vote (see 4.1.4).

4.4 Election administrator use cases

The following use cases illustrate the operations realized by the election administrator. In these scenarios we will again refer to election X, conducted by election administrator Alice.

[12] If fact, nothing prevents the election trustee client and the auditor client from being the same software application in different modes of operation.

4.4.1 Election creation

The following describes the process of creating a new election. Election set-up goes through four steps. The first is election creation, where the basic parameters of the election are defined. Following that, the election commission (4.4.2), candidates (4.4.3) and voters (4.4.4) for the election must be selected, reviewed and finalized.

An optional usability feature, which could make election creation easier, is the concept of election templates. These would be nothing more than snapshots of election configurations, which can be saved and loaded, so that said configuration can be reused in other elections without filling the fields manually every time.

1. Alice logs into the Plone server hosting PloneVote and goes into a public folder were elections are allowed as content.

2. Alice adds a new election to that folder (in this case, an SCE Election). This requires the appropriate Plone security privileges.

3. Alice then goes through a few screens configuring the basis of the election:

   (a) The first section allows her to give the name and a brief description of the election, as well as selecting the ballot scheme and vote counting scheme to be used. Some options for the ballot scheme may also be available here, such as if a blank vote is allowed, if candidacy confirmation is required or a minimum and maximum number of candidates to select. If election templates are implemented, this also allows Alice to populate all election configuration with a previously saved template.

   (b) The second section sets the security settings, including: number of election trustees, number of election trustees needed to decrypt the election and minimum number of verifiable shuffles required before decryption. Any other election-dependent parameter that affects the Election Security Protocol (3.2.4) should be set here as well.

   (c) The third section allows Alice to look at the election Phases and steps, and set timed transitions for those steps that require being available for a set period of time (e.g. list reviews, voting, verifiable shuffling). The timers for the transitions are specified, together with whether or not a manual transition can be triggered by the election administrator as an override.

   (d) The fourth section asks for a page object for the election to be created as a publicly visible announcement of the election. This page will be displayed as the default view of the election, with added links to actions such as viewing the voter or candidate lists, voting or seeing the results, depending on the election state. If other messages for the election can be customized, they may either appear as part of this section or as a fifth section immediately following.

(e) Finally, if election templates are implemented, Alice is given the option of saving all the previously selected configuration as a named election template (e.g. "Chairman Election").

4. After the election has been created, Alice is shown the election page, where options for selecting the election commission, creating the candidate list and creating the voter list, are offered to her (no other user role is given these options). She may perform this steps either before or after publishing the election, but potential candidates, voters and commission members will only be notified after the election has been published[13].

5. Alice publishes the election as she would any other Plone content.

4.4.2 Election commission selection

This describes the process followed by Alice (as the election administrator) to select the election commission and confirm their membership acceptance.

1. Alice goes to the election's public page and selects the option for viewing the election commission.

2. If she has not yet set-up any commission members, a page is shown asking her for a number of users equal the number of trustees indicated at election creation time (see 4.4.1). She must select a Plone user for each election trustee and confirm her selections with a submit action.

3. After the election has been published and the list of trustees confirmed, all users listed as trustees will be sent a notification by the system (site and/or mail) and allowed to accept or decline the position.

   (a) Accepting and rejecting trustees should be immediately marked as so in the election commission page for election X, which is visible only to the election administrator before being finalized.

   (b) Alice can always change the user for any trustee that has not accepted the role, which will cause the system to notify both the old and the new trustees of the change. In this prototype of the system, it may or may not be possible to remove accepting trustees. No changes to the list of trustees may be done after the Key Set-up step of the SCE (step 3) has ended.

   (c) If any named trustee declines, Alice gets a message immediately (site and/or mail) notifying her of the fact. The message should provide a link to the election commission selection page for election X. She can there substitute all rejecting trustees for new users who will get immediately notified and asked for their acceptance of the role.

[13] In other words, review for all three lists cannot commence until after the election has gone public.

(d) As soon as all trustees selected have been marked as having accepted, Alice gets a notification (site and/or mail) of the fact.

4. Trustees work on the election's public key set-up as described in use case 4.3.2. After the public key has been created by the server from all uploaded partial public keys, the election administrator gets another message informing her of the fact.

4.4.3 Candidate list creation, review and finalization

The following use case describes how Alice creates, edits and finalizes the candidate list.

1. Alice goes to the election's public page and selects the option for the candidate list.

2. The user list selection tool opens, using this tool:

   (a) Alice sets a series of criteria specifying who should be allowed to be a candidate in the election. This uses Plone user and group metadata and may make use of (for example) properties defined by the Faculty/Staff Directory product[4].

   (b) She gets a list of all users matching that criteria, where she can remove or add other Plone users to adjust the list. She also fills a small text note justifying the change.

   (c) For candidates, she is also allowed to manually add options which are not Plone users, by giving a name and a picture.

3. When Alice has set-up the candidate list exactly as she wants it, she marks it for review. After the candidate list is marked for review and the election has been published, this list is shown as public, together with the criterion, manual modifications and justifying notes. Candidates are notified and may accept or reject their candidacy as in 4.2.1.

4. Alice may modify the list to respond to notifications and complains from potential candidates and other users, simply by going to the list and manually deleting or adding candidates.

5. After either a time interval or a manual action from Alice, the candidate list is declared as finalized. This gets the list published to all and makes the server reject any further changes to it. If candidate confirmation is required for the election, candidates not marked as having accepted their candidacy do not appear on the final list.

4.4.4 Voter list creation, review and finalization

The following use case describes how Alice creates, edits and finalizes the voter list.

1. Alice goes to the election's public page and selects the option for the voter list.
2. The user list selection tool opens, using this tool[14]:
   (a) Alice sets a series of criteria specifying who should be allowed to vote in the election. This uses Plone user and group metadata and may make use of (for example) properties defined by the Faculty/Staff Directory product[4].
   (b) She gets a list of all users matching that criteria, where she can remove or add other Plone users to adjust the list. She also fills a small text note justifying the change.
3. When Alice has set-up the voter list exactly as she wants it, she marks it for review. After the voter list is marked for review and the election has been published, this list is shown as public, together with the criterion, manual modifications and justifying notes. Listed voters are notified and may review the list as in 4.1.1, while users not included in the voter list may follow use case 4.1.2 to request being listed or simply review the list.
4. Alice may modify the list to respond to notifications and complains from potential voters and other users, simply by going to the list and manually deleting or adding voters.
5. After either a time interval or a manual action from Alice, the voter list is declared as finalized. This gets the list published to all and makes the server reject any further changes to it.

4.4.5 Phase and step transition manual triggering

For those steps and phase transitions which are configured at election creation time as allowed to be triggered by manual action, the election administrator can cause the step or phase transition to take place in a single step:

- For most step and phase transitions, the election page will show the election administrator one-click actions for performing any transition available in the current election state.
- For step transitions dealing only with the trustee, candidate or voter lists, the actions will instead be made available in the page for the list itself.

[14] Unlike for candidates, there cannot be added entries to the voter list that do not correspond to a Plone user, for the simple reason that the system allows only authenticated Plone users to cast votes. Maybe a future version could support adding non-Plone users with custom means of authentication (OpenID, etc).

This actions may be implemented as either links, buttons or Plone workflow actions. Confirmation should be asked from the user when optional but expected actions for that step have not yet taken place, or when the action was also programmed with a time trigger and the time for the transition has not yet been reached.

4.5 Auditor use cases

The following use cases follow election auditor Edna in determining the accuracy of election X, created by Alice. Edna may or may not be a voter, candidate or election trustee as well. She trusts the PloneVote Auditor Client (and that at least a large proportion of voters will verify their own vote as in use case 4.1.4), but not necessarily the code running on the server or the members of the election commission.

The PloneVote Auditor Client is a desktop application or similar client code that can be verified by any third party with experience in programming and who understands the Election Security Protocol (3.2.4) and the cryptographic principles on which it is supported. The auditor client and the election trustee client could very well be the same program or share code.

4.5.1 Verifiably shuffling the votes

During the vote counting phase, but before vote counting takes place, Edna may increase the privacy of the election by verifiably shuffling the votes.

1. Edna opens the auditor client, points it to the correct PloneVote site by providing the url and selects the election she wishes to verifiably shuffle. Logging-in is not needed for this use case.

2. If the election is in the correct state (step 14 of the SCE), then the client offers the option to perform a verifiable shuffle.

3. If another client is already performing a shuffle, Edna will be notified of this and asked if she wants to proceed or wait until the other shuffle has been performed. For shuffles occurring at the same time over the same set of ballots, the server will accept as valid only the first shuffle it receives, rejecting the other with an appropriate notification[15].

4. Edna selects the option to perform a verifiable shuffle and waits, the client should show her a progress bar and a description of the steps it is performing, but should be able to perform the shuffle without further input. The process involves:

[15] This is because parallel shuffles add no privacy, and may in fact harm it if they lead to shorter chains of shuffles. A single serial chain of shuffles is all that the server must accept. Note, however, that the approach taken may cause problems with DOS attacks performed as persistent shuffling by a single corrupt party, which the current prototype does not defend against. Those would be very visible attacks and threaten only the privacy, but not the accuracy of the election. The attack also requires that the attacker wins all races against honest auditors performing their own shuffles.

(a) Notifying the server that is starting a verifiable shuffle, so that other auditors attempting to do so at the same time will be warned.

(b) Downloading the latest set of encrypted ballots.

(c) Performing the verifiable shuffle cryptographic process (mixnets, see [1, 3])

(d) Uploading the new shuffled encrypted ballot set, together with the proofs of correct shuffling.

(e) Awaiting for the server to accept the verifiable shuffle and notify the client back. The server should verify the proofs before accepting the new ballot set.

(f) Checking that the latest set of encrypted ballots is now the uploaded set.

5. The auditor client notifies Edna that the process has been completed and whether or not it was successful. She closes the client or tries again.

4.5.2 Election verification

After the votes have been counted, Edna may use the auditor client to verify the result of the whole election.

1. She opens the auditor client, points it to the correct PloneVote site by providing the url and selects the election she wishes to verify. Logging-in is not needed for this use case.

2. If the election is in the correct state (step 17 of the SCE), then the client offers the option to verify the election.

3. The client first downloads all published information required for verification: ballot information, voting receipt and voter lists, all encrypted ballot sets, the final decrypted ballot set and all cryptographic proofs of shuffling/decryption. After this point, it no longer needs to have any connection with the server to verify the results.

4. The client asks Edna whether or not she voted on the election.

   (a) If she did, she is asked to enter her voting receipt so that the client may check that it is included in the published receipt list. If not, the auditor client warns her that her vote may have been deleted.

   (b) If she did not, she is asked to check the list of voters and verify that she does not appear in it. If she does, the auditor client warns her that someone may have impersonated her to vote.

5. Then, if no error occurred checking Edna's vote, the auditor client displays a progress bar and step indicator as it performs the following steps:

(a) Do some sanity checks over the data sent to it by the server (e.g. the number of counted votes corresponds to the number of voters listed as having voted and the number of voting receipts).

(b) Verify that every vote in the first encrypted set corresponds correctly to one of the published voting receipts.

(c) Verify all shuffles from the first ballot set up to the one who was finally decrypted.

(d) Verify the proofs of decryption (partial decryptions and their combination).

(e) Finally, do a recount of the votes and check that it coincides with the published results.

6. If any step of the verification failed, the client warns of potential fraud in the election and shows some information about the problem. If not, it shows a notification saying that the election appears to have been counted correctly and shows the user the recalculated results, urging her to verify that they are the same as those published on the PloneVote site.

5 Features and priorities

5.1 About priorities

In the following section, we give a list of the system desired features for each of its mayor components, along with an associated priority level for their implementation on the current prototype. Before doing so, we list the contemplated priorities and their meanings:

- **P0** - Architectural. Must be implemented for the current prototype. Features marked P0 are part of the core responsibilities of PloneVote and are necessary for the system to be implemented at all. A prototype missing a P0 feature will not run or else will not be able to address the basic problem PloneVote is intended to solve in any satisfactory manner. A prototype implementing only all P0 features should be enough, in theory, to act as a proof of concept of the system when manipulated expertly (perhaps at a level as low as using a python terminal and modifying Zope/Plone objects directly)

- **P1A** - Basic usage. Features marked P1A are those necessary to support the use cases we are more interested in or require for the current prototype. A prototype missing P1A features will not be usable directly to conduct a full election, even if the core components and architecture are in place, at best it would require painful workarounds to be used.

- **P1B** - Basic verification. Features marked P1B are required to have an election that supports all the verification characteristics of our Election

Security Protocol (for example, having an auditor client and encrypting votes in JavaScript client code). A system missing P1B features may be usable but cannot be fully trusted or may require unreasonably complex manual verification to become trustworthy.

- **P2** - Significant usability or auditability features. These features are not required to effectively use and trust the PloneVote prototype, but may present significant advantages in: ease of use, ease of administration, versatility, etc, and should be seriously considered for inclusion even on the first prototype.
- **P3** - Interesting features for the first prototype. Other features that could be good to have for the first prototype and offer some significant advantage compared to a system which does not implement them, but are ultimately not very important.
- **P4** - Cosmetic features. Low importance, simple to implement features that still could be included in the first prototype if time allows.
- **P5** - Future. These features are explicitly not intended for inclusion in the first PloneVote prototype, but could be contemplated in future prototypes.

Minimally, features at P0 and P1 (A and B) should be implemented for the system to be considered ready for use.

5.2 Feature listings

5.2.1 Election Security Protocol : Implemented algorithms and functions

The following features relate to the algorithms and primitive functions that must be implemented in different points of the system to provide the cryptographic building blocks of the Election Security Protocol. Where not otherwise specified, this functions are to be implemented as part of one or many python libraries that can be used by both the server (within Plone) and the trustee or auditor client (outside of Plone).

Priority	Feature	Description
P0	Private and public key generation	The system should provide the required functions to generate a key-pair of private and partial public key for each trustee.
P0	Threshold encryption set-up	The system should provide the required functions to construct the election public key from the partial public keys, following a threshold encryption scheme in which actions from k of n holders of private keys are required for decryption.

Priority	Feature	Description
P1B	Threshold encryption public key verification.	The system should provide the required functions to verify, given the set of n partial public keys, that the public key resulting from a k of n threshold encryption with those keys was correctly generated.
P0	Public key encryption (any form)	The system should allow for an arbitrary text to be encrypted with the combined public key for the threshold encryption scheme. This may be implemented in python and/or JavaScript code.
P1B	JavaScript public key encryption	Same as above, but requiring a JavaScript implementation for in-browser client encryption.
P1B	Vote receipt generation	The system should provide the required functions (in JavaScript code) to make a hash of the encrypted vote to use as a vote receipt. Since generating a vote receipt is done by means of a hash, this feature doubles as allowing vote receipt verification.
P0	Partial decryption creation	The system should provide the required functions to create a partial decryption of an arbitrary encrypted text, using the private key corresponding to one of the partial public keys of the threshold encryption scheme.
P0	Partial decryption combination	The system should provide the required functions to combine k partial decryptions (on a k of n threshold encryption scheme) to create a full decryption of an arbitrary encrypted text to its corresponding plaintext.
P1B	Decryption proofs	The functions used for partial decryption creation and combination should be verifiable. In particular, partial decryptions should offer proofs of correct decryption.
P1B	Decryption verification	The system should provide the required functions to verify that decryption proofs are correct, ensuring that partial description creation and combination steps were all performed honestly.
P0	Verifiable shuffling	The system should provide the required functions to verifiably shuffle a set of votes (ie. a mixnet), generating a non-interactive proof of correct shuffling and without associating elements from the starting set to those of the shuffled set.
P0[16]	Shuffle verification	The system should provide the required functions to verify the correctness of a verifiable shuffle, given the starting set, the shuffled set and the generated proof of correct shuffling.

[16]This is P0 and not P1B, because the server needs to verify shuffles in order to prevent unprivileged users from uploading arbitrary ballot sets as shuffles, which would deny all accuracy

Priority	Feature	Description
P3	Vote encryption auditing	The system should provide the required functions in JavaScript and python to obtain and verify a proof of the fact that the vote was encrypted honestly by the JavaScript voting client, without the need for verifying its code (trusting a python program or similar desktop client code instead).

5.2.2 Ballot and counting schemes

The following features all deal with the underlying infrastructure required for vote casting, counting and count verification. The first scheme that we wish to support, multivote with multivote counting, is separated into various features to adequately express the priority of supporting each characteristic of the scheme. Conversely, support for preferential voting is divided only into basic support for casting preferential votes and support for counting them, as support for preferential voting as a whole is a low priority feature.

Priority	Feature	Description
P0	General ballot and counting mechanism interfaces and base support	The system should provide the required interfaces, classes and methods to represent stored ballots, selectable voting schemes and selectable vote counting schemes and their valid combinations and configuration arguments. That is, the machinery required to create ballot scheme and vote counting scheme extensions for PloneVote, should be supported.
P0	Multivote ballot storage and representation	The system should provide a multivote ballot scheme extension, with the ability to define, represent and store multivotes within the Plone instance.
P1B	Multivote limits enforcement and ballot verification	The system should allow for the multivote scheme of an election to have defined parameters *min* and *max*. It should also provide the functions to detect whether an stored (decrypted) ballot is valid or invalid.
P1A	Candidate list definition	The system should provide an object to describe the list of candidates for an election and their properties.
P1A	Candidate list assignation to ballot	The system should allow a multivote ballot template for an election to be created from the ballot scheme configuration for said election and a list of candidates.

for the election.

Priority	Feature	Description
P1A	JavaScript vote casting client	The system should allow for multivotes to be created inside a vote casting JavaScript client. The client should provide an user friendly, graphical way of checking and unchecking candidates and verifying the correctness of the vote to be cast. Those votes should then be encrypted (if the corresponding feature is also implemented) and submitted to the server.
P1B	Multivote vote counting process	The system should provide a vote counting scheme extension to adequately count a multivote election. This process should work both inside the server or from a desktop client.
P2	Multivote support for blank votes	The system should allow for the multivote scheme to be configured with an optional per-election flag, enabling or disabling blank votes. The vote casting client should be aware of this option, as should the vote counting process.
P3	Preferential ballot scheme	The system should allow for preferential ballots to be cast through a JavaScript client, verified in client and server, stored and in general used for an election. Blank vote should be supported.
P3	Preferential ballot counting: Schulze-STV	The system should provide the Schulze-STV vote counting scheme extension to adequately count a preferential ballot election. This process should work both inside the server or from a desktop client.
P4	Preferential ballot counting: others	To test the ability of the system to provide different vote counting schemes for the same ballot scheme, other counting algorithms for preferential ballots could be implemented. Borda Count, for example.

5.2.3 Candidate and voter list creation tools

The following features relate to the core tools used for creating, reviewing and editing candidate and voter lists, which will be part of the PloneVote system. These tools may be either tightly or loosely coupled to the main SCE module in implementation, but are described separately because their concerns are different: create and maintain lists of users, rather than configure and run the details of an election process.

Priority	Feature	Description

Priority	Feature	Description
P1A	Constrain based candidate/voter list creation	The system provides a tool for automatically creating a list of candidates or list of voters by selecting a series of constrains over the metadata of Plone users, including membrane and Faculty/Staff directory information, and compiling all users who satisfy the constrains. After list creation, no users are added or deleted automatically, even if they change in such a way that they now fulfill or no longer fulfill the initial constrains.
P1A	Manual candidate/voter list modification	The candidate/voter list tool allows the election administrator to manually add or remove users from the list.
P2	Auditable modification	Brief annotations are required to justify manual enlisting or delisting from a candidate/voter list. The system logs all changes and the initial criterion, as well as the users who satisfied it at list creation time, so that the lists can be audited.
P3	Non-user candidate entries	Candidate lists support adding entires that do not correspond to Plone users.
P1B	Candidate/voter list finalization and publication support	By either timed step change or manual action from the election administrator, the candidate/voter list can become finalized. The server prevents further modification of finalized lists. A finalized user list is usable by the ballot scheme extensions, to populate candidate ballots, and by the SCE system, to determine an user's right to vote on in election.
P2	Voter/candidate notifications	The system should notify voters and candidates by site message and/or e-mail whenever they are added or deleted from a voter or candidate list, including when the list is originally created. The system should also notify of list finalization. Configuration options for the election to suppress notification should be considered.

Priority	Feature	Description
P2	Candidature confirmation	It should be possible to configure the system to require candidature confirmation from users appearing on a candidate list for a particular election. If the election is configured to require candidature confirmation, candidates on the initial candidate list who do not confirm their candidature will be deleted from the list before list finalization. As part of this feature, the system should implement appropriate notifications and means of accepting candidature. It should also mark users in the candidate list as having accepted their candidature or not, accordingly.
P3	System supported candidate/voter list complain mechanism	The system should allow users appearing in candidate/voter list to request delisting to the election administrator, and users not appearing to request enlisting. Sending and honoring this requests should be a simple action within the system and notifications should be implemented appropriately.
P3	Candidate profile support	Candidates should be allowed to add a profile (a linked Plone or external web-page) to their entries in the candidate list, which would be shown together with either the election announcement page, the finalized candidate list and/or the voting ballot.
P5	Constrain fulfillment change notification	The system could notify the election administrator if, during the candidate/voter review step of the election, a user who previously did not fulfill the constrains to be automatically included in the list, suddenly does. The notification should allow the administrator to manually add the user to the list. A converse mechanism could be provided for users who suddenly no longer fulfill list constrains.

5.2.4 Standard candidate election server-side actions support

The following features refer to the services required for running SCEs within the server and all related web user interfaces for the PloneVote system, particularly those used to configure, view or vote in the election. However, the server-side actions that are triggered only in response or in collaboration with the trustee or auditor client programs are included in the next section, such as decryption, counting and proofs publication.

Priority	Feature	Description

Priority	Feature	Description
P1B	SCE Scheduling core	The system should implement the basic mechanisms to run an election as per the SCE model. This includes keeping track of time triggered and manual step transitions, as well as a general mechanism for steps to define pre and post conditions. This also includes important steps that are realized by the server alone, without interaction with any other user, such as the vote phase set-up and server-side shuffle steps.
P1A	Minimal election creation configuration	The necessary web side graphic interface, screens and commands to create an election should be provided, as described in use case 4.4.1. It should be possible to configure at least the following parameters of an election: name, description, ballot scheme, vote counting scheme, Election Security Protocol settings, type (timed vs manual) of step transition triggers and corresponding times.
P2	Ballot and voting scheme configuration	The SCE module should be able to query the selected ballot and voting scheme modules for the election and present their configuration options to the election administrator along with the standard election configuration options.
P2	Customized election page support	The system should allow a web page for the election to be created, associated with it, published with the election in Plone and integrated with the election process. Said integration includes linking the election page in notifications and adding action links to said page based on the election state and viewing user's role. The layout of most of the page and its contents should be mostly free form.
P3	Election messages configuration	The system may support customizing the text of certain server messages and notifications on a per-election basis.
P3	Election templates	The system may support election templates, which allow configuration for one election to be saved and reused for another.
P1B	Election commission selection and review	The system allows the election administrator to realize all steps from use case 4.4.2. In particular, selecting, adding and deleting members of the election commission, as well as monitoring their acceptance or rejection of the role.

Priority	Feature	Description
P1A	Candidate/voter list integration	The system integrates the candidate and voter lists of section 5.2.3 with the SCE, such that use cases 4.4.3 and 4.4.4 are supported to the extent in which the corresponding features are supported for candidate and voter lists.
P1A	Integrated voting JavaScript client	(See 5.2.2) The JavaScript vote creation and encryption client should be integrated with the SCE process, being run immediately after the user selects the action to vote and having the corresponding vote uploaded to the server.
P1B	Server-side voter confirmation and tracking	The server should keep track of which users are allowed to vote, which have already voted and which are left to vote. However, it should never associate a voter with their cast vote, just keep track of who has voted. The server should make sure that only correctly authenticated users can vote[17].
P5	External system voter authentication	For future versions, the system may allow to authenticate voters by means other than a Plone login. This would require changes to the user lists implementation, as well as to the server-side voting authentication and authorization process.
P1A	Election results report page	After election decryption and count, the server should automatically generate a report page with the results for the election and a link allowing each user to verify their own vote.
P2	Election results notification	Site and/or e-mail notifications should be send to all voters and candidates after the results of the election have been published.
P1B	Own vote verification page	The server should provide a link from the results page to a page showing the list of people having voted and a separate list of voting receipts for all cast votes, together with appropriate instructions so that any user can verify that their own vote was cast correctly.
P4	Visual hash receipts	The system may support a visual representation (e.g. color bars) of hashes and show such representation for each vote receipt given to an user or shown in the vote verification page. This would simplify visual verification of the receipt presence in the vote receipt list.

[17] Illegally added votes can be detected at verification time, but they invalidate the election, so it is in our best interests that an honest server does not allow adding illegal votes.

5.2.5 Trustee/Auditor client

The following features are either those which need to be implemented in the trustee or auditor software clients, or those that should be implemented on the server side to allow tasks that are done in concert with client code (other than the voter JavaScript client). At the beginning of each feature description, we have marked if it relates to the trustee client, the auditor client of both. If both clients are implemented as the same software, then that software should, of course, implemented both sets of features.

Priority	Feature	Description
P1A	Basic client to server connection	[Both] The client program should be able to communicate with the server after being provided its url. After connection, the client should be able to transparently download all the public files from the server it may require to operate, including: results, ballot sets, proofs, lists, etc. The client should also be able to upload files to the server which are first checked and then processed by specifically called code inside the server-side PloneVote system (for example, a verifiable shuffle).
P1A	Graphical operation	[Both] The client should work by means of an intuitive and easy to use graphical user interface (GUI), for all tasks.
P2	Client log-in	[Both] The client program should be capable of creating an authenticated session with the Plone server in which the user logs-in to their account on the server in a secure way, by providing their Plone user name and password to the client. After client log-in, actions performed through the client software can safely be considered by the server as being done on behalf of the authenticated user and not a third party. This is required, for example, for trustee key registration. If this feature is not implemented, certain actions (in particular key set-up) will instead require the user to create files through the client software and then upload them from the web interface to the server.
P1A	Election listing	[Both] The client program should, upon connecting to a PloneVote server, show the list of currently running elections. After a particular election is selected, the client must check the election state and display a list of possible actions that may be currently performed on that election using said client.

Priority	Feature	Description
P1A	Election key-pair creation	[Trustee] The client should be able to create a private and partial public key pair for the PloneVote Election Security Protocol. After creation, both keys are to be saved in the machine where the client is running. It should be possible to easily extract the partial public key as a file and upload it to the server, either manually or automatically.
P3	Passphrase protected private key	[Trustee] The client may offer the user the option of passphrase protect their private key, either with their Plone log-in password or with a newly created passphrase. The passphrase will thereafter be need to use the private key, as it will be encrypted with a symmetric key scheme using the passphrase.
P3	Save private key to file	[Trustee] The client may offer the user the option of saving their private key to a portable file. If this option is provided, it is also recommended that passphrase protected private keys be supported as well, and that the client prompts the user for a passphrase to protect the key before saving it to an arbitrary file.
P5	TPM/ Cryptoprocessor private key storage	[Trustee] The client could add some security to non-portable private keys stored in the same computer as the client itself, by using hardware key management, when it is available, to tie the private key file to the current machine. This is potentially a very complex feature to implement, and asking election trustees to use full drive or filesystem encryption accomplishes the same goal.
P2	Election partial public key automatic upload	[Trustee] If client log-in is supported, the client should be able to automatically upload the partial public key for an election directly to the server after log-in. The option to do key set-up should be available from the client if the election is on step 3 of SCE. The user should be prompted to perform key-pair creation if she has not done so already.
P1B	Election threshold scheme set-up verification	[Trustee] During step 4 of an SCE, the client should show the option to verify the threshold scheme set-up. The client needs not log-in or authenticate with the server for this operation, but may need to ask the user if she is a trustee in the election and to select the public and private key pair she is using for that election from a list of stored key pairs[18].

[18] Alternatively, the client could be made to always use the same key-pair for all elections on the same server or remember the key-pair used for each election, if key set-up is done via

Priority	Feature	Description
P1B	Verifiable ballot shuffling	[Auditor] During step 14 of an SCE, the client should show the option to perform verifiable ballot shuffling. This should work as in use case 4.5.1.
P2	Shuffle completion test	[Auditor] After verifiably shuffling the ballots, uploading them with proof to the server and being notified by the server that the new ballot set was accepted, the client should re-start the verifiable ballot shuffling process, up to the point in which it has access to the latest ballot set. At this point, instead of continuing the shuffle, it should merely confirm that the chain of shuffled ballot sets includes the one it just uploaded after the previously last one.
P1A[19]	Partial decryption creation	[Trustee] During step 15 of an SCE, the client should show the option to upload a partial decryption of the votes. If such option is selected, the client should download the last shuffled ballot set (and perhaps check that enough shuffles took place since the original ballot set) and create a partial decryption of the last ballot set, uploading it to the server.
P1A	Server-side decryption	[Trustee] As soon as the k partial decryptions for the latest shuffled ballot set which are required for the threshold encryption scheme have accumulated on the server, the PloneVote system will decrypt said ballot set and proceed to vote counting.
P1B	Election shuffle chain and decryption verification	[Auditor] During step 17 of an SCE, the auditor client should provide the option to audit the election. It should be able to check all verifiable shuffles, and the proof of decryption, as well as performing sanity checks on all data given by the server about the election.
P2	Auditor client own vote verification	[Auditor] The auditor client should also provide the ability to verify the voters own vote, in a manner redundant with the web site page (see use case 4.5.2). The client should also contrast the number of decrypted ballots with the number of users having voted and the number of user receipts.

automatic upload of the partial public key.

[19]Being able to do a partial decryption somehow (e.g. directly invoking the correct python code) is fundamental for the system (e.g. P0). Doing so through a graphical client software is a basic usability feature.

Priority	Feature	Description
P2	Auditor client recount	[Auditor] The auditor client should perform an independent recount of the decrypted votes as well and show the results to the user, in order to ensure that they coincide with the results published by the server[20].
P5	Desktop client voting	[Other] For future versions of the PloneVote system, a full desktop client for voting may be provided, in addition to the JavaScript vote casting client.

5.2.6 Mixed features

Priority	Feature	Description
P1A	Language support: English	All user facing messages, notification and interface elements should be available in English, and shown in that language if the Plone site and user system are both configured to English locales.
P1A	Language support: Spanish	All user facing messages, notification and interface elements should be available in Spanish, and shown in that language if the Plone site and user system are both configured to Spanish locales.
P2	Event log of all election actions	The server should keep a log of all election actions, including, at least, those described in section 3.4.4. This in order to facilitate both diagnostics and a review of the election in case of verification errors or suspected anomalies.
P3	Append only log file	The server log file for all information related to a PloneVote election should be append only and not overwritable by the Plone instance or the PloneVote system. Ideally, the log should be done to write-once media that cannot be physically altered by the server after initial writing.

References

[1] Josh Benaloh. Simple Verifiable Elections. In *EVT '06, Proceedings of the First Usenix/ACCURATE Electronic Voting Technology Workshop,* August 1st 2006, Vancouver, BC,Canada., 2006. Available on- line at http://www.usenix.org/events/evt06/tech/.

[20] Note that without this feature and the previous one, some manual auditing of the files retrieved by the auditor client may be required in order to ensure that the server is not providing it different information that the one it is showing on the Plone site.

[2] Ben Adida. Helios: Web-based Open-Audit Voting. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association. Available on- line at www.usenix.org/events/sec08/tech/.

[3] Ran Canetti and Ron Rivest. 6.879 Special Topics in Cryptography, Lecture Notes 19 and 20. MIT Course, Spring of 2004. Available on-line at http://courses.csail.mit.edu/6.897/spring04/materials.html

[4] http://plone.org/products/faculty-staff-directory