Practical Approach to Anonymity in Large Scale Electronic Voting Schemes

Andreu Riera, Joan Borrell
Departament d'Informàtica — Universitat Autònoma de Barcelona
Edifici C — 08193 Bellaterra — Catalonia (Spain)
E-mail: {ariera,jborrell}@ccd.uab.es

Abstract

Anonymity of ballots in electronic voting schemes usually relies on the existence of some kind of anonymous channel between voters and ballot collecting authorities. Currently, there exist solutions based on the mix concept, which allow for anonymous e-mail communications. However, integration of such solutions into the implementation of a voting scheme has some problems. In this paper we propose a large scale voting scheme based on the concurrent operation of multiple electronic electoral colleges, with no need for independent anonymous channels. Anonymity of ballots is assured by shuffling of ballot boxes by a set of mobile agents acting on behalf of a central voting authority. Our voting scheme is totally self-contained and suitable to be implemented over large internets.

1. Introduction

The objective of electronic voting schemes is to allow elections to take place securely over general-purpose and open computer networks. During the ballot collecting process, a set of eligible voters use the computer network to cast their ballots. After some time, the system stops accepting ballots. The counting process is initiated and, finally, the tally is published.

Different security requirements for voting schemes have been proposed. The following list includes the most widely accepted ones [2, 7]:

Accuracy: A voting scheme is accurate if (1) it is not possible for a validated ballot to be altered, (2) it is not possible for a validated ballot to be eliminated from the final tally, and (3) it is not possible for an invalid ballot to be counted in the final tally.

Democracy: A voting scheme is democratic if (1) it permits only eligible voters to vote, and (2) each eligible voter can vote only once.

Privacy: A voting scheme is private if (1) neither ballot collecting authorities nor anyone else can link any ballot to the voter who has cast it, (2) no voter can prove that he or she voted in a particular way, and (3) all ballots remain secret while the voting is not completed.

Verifiability: A voting scheme is verifiable if voters can independently verify that their ballots have been counted correctly.

The first privacy property, known as anonymity, is probably the cornerstone of secure voting schemes. There are no obvious solutions to anonymity, since in order to preserve the democracy requirement, voting authorities responsible for collecting ballots should have assurance of the identity of voters who contact them. The most widely accepted solution to this problem found in the literature consists of assuming the existence of an anonymous communication channel. A voter casts his or her ballot in two sessions. The objective of the (non-anonymous) first session is to assure the democracy requirement. This is achieved by authenticating the voter and providing him or her with some sort of one-time voting token or authorization. Some cryptographic mechanism is usually involved in this step to prevent voting authorities from knowing exactly which authorization each voter has received. The second session consists of using the anonymous channel to cast the desired vote, together with the authorization, to the appropriate ballot collecting authority.

Most references on electronic voting schemes do not specify how the required anonymous channel can be implemented. Nonetheless, anonymous channels are normally implemented by a sequence of mixes (a mix-net). The concept of a mix was first introduced in [3]. Even though current implementations of this concept can realistically be used for anonymous e-mail communications, they have disadvantages (listed in Section 2) when used as anonymous channel between voters and ballot collecting authorities in voting schemes.

This paper presents a voting scheme based on the concept of electronic Electoral College (EC) introduced in [13]. An EC is responsible for collecting ballots from a reduced set of registered voters. By coordinating a number of concurrent ECs, the voting scheme is scalable and therefore it may be used to perform large scale elections over large internets. Moreover, the voting scheme does not need any mix-net or any other means of independent anonymous channel. Only the usual communication facilities of networks are required. Voters cast the desired ballots, specially enveloped, to the respective ECs during a single non-anonymous voting session. In this way, every EC accumulates enveloped ballots in its own ballot box. The desired anonymity is provided by shuffling ballot boxes a number of times. The process of shuffling ballot boxes is implemented at the end of the election by a set of mobile agents. See [5] for an introduction to mobile agents and a description of a possible security infrastructure to operate them.

Our proposal represents a concrete solution to the anonymity of electronic voting schemes which does not rely on any external mixes. Only the components of the voting system themselves are needed. Still, anonymity of ballots is provided through the same functionality which constitutes the essence of mix-nets. In particular, the processing of ballots described in this paper is strongly based on the mixing techniques presented in [8]. The results of this paper allow the development of an anonymous voting scheme totally self-contained, thus solving one of the currently most important issues concerning practical implementation of secure voting schemes. In addition, all disadvantages inherent to the use of mix-nets in voting schemes are removed.

The rest of the paper is structured as follows. In Section 2, the concept of a mix-net is reviewed, and the disadvantages of its use in electronic voting schemes are listed. Section 3 is devoted to the operation of our voting scheme. The set of voting authorities and the procedures performed during each of the election phases are described. In Section 4 we discuss how the proposed scheme assures anonymity. The fulfillment of all other security requirements is discussed in Section 5. Finally, Section 6 contains the conclusions of the paper.

2. Use of mix-nets in voting schemes

Implementing an anonymous channel in a computer network is not a straightforward matter. Communication protocols include headers in each datagram which contain both source and destination addresses. Among the proposals for anonymous channels found in the literature, the most suitable to be implemented are those in [3] (the first paper dealing with anonymous communications) and [12]. Nonetheless, a successful attack against [12] is described in [11]. There are possible countermeasures, but they actually make the proposal equivalent to that in [3]. As a consequence, anonymous channels are normally implemented in computer networks by using the concept of a mix described in [3], or slight variations of it (see for example [8]).

A mix is an entity that, in addition to forwarding incoming messages, hides the relationship between incoming and outgoing messages. This is done by grouping together a number of incoming messages, replacing the original source addresses by the address of the mix itself, shuffling the messages and, finally, forwarding them to the intended recipients. Both the contents and the destination address of any message sent to a mix are encrypted. Decryption has to be possible only by the mix. To prevent correlation between encrypted incoming messages and decrypted outgoing messages, successive encryptions of the same message should give different results. It is also necessary that all messages have uniform length. Otherwise, every outgoing message could be correlated to the incoming message(s) of the same length.

Using a mix as a common message forwarder for a group of users assures unlinkability of senders and receivers to anyone except to the mix itself. To avoid having to trust a single entity, several mixes are arranged in a sequence, thus constructing a mix-net as depicted in Figure 1. Messages are processed by each mix, in turn. In a mix-net, a single mix is required to be honest in order to prevent other mixes from correlating incoming and outgoing messages to/from the mix-net. Still, if all mixes collude, the origin of every outgoing message can be traced back.

Currently, implementations of the mix concept are available on the Internet [8, 6], but they are oriented to e-mail communications. Furthermore, the use of mix-nets in voting schemes has some disadvantages:

- Delivery of ballots to ballot collecting authorities is performed during the voting process, so that the third privacy property cannot be properly assured.
- Two different sessions are required to cast a ballot. One is to obtain a voting authorization from a

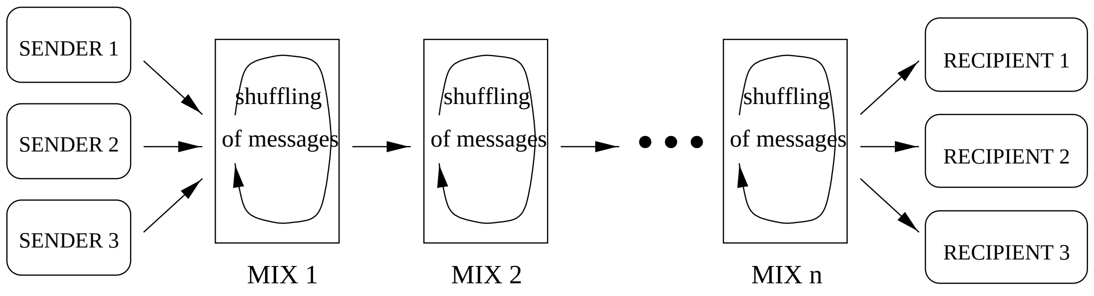

Figure 1: **Schematic diagram of a mix-net architecture.**

certain voting authority, and the other to cast the ballot through the mix-net.

- Malicious voters can obtain valid authorizations and then skip casting their ballots. When the tally is published, they can claim that the voting system has removed validated ballots.

- Under low traffic conditions (i.e. at moments when few ballots are cast), mixes introduce random-looking decoy ballots. However, ballot collecting authorities are able to differentiate between valid ballots and decoy ballots. Therefore, this mechanism does not assure anonymity if ballot collecting authorities are undertaking traffic analysis tasks. In addition, it increases the load on the system.

- The whole voting system relies very much on a set of external entities.

3. Operation of the voting scheme. Election phases

Operation of our voting scheme is based on the hierarchical arrangement of voting authorities presented in [13]. A global Election Authority (EA) of permanent nature is situated at the root of the hierarchy. The EA is in charge of the whole election and it is responsible for the creation and maintenance of the electoral roll. Actually, it represents the electronic version of the official authority in charge of elections in most countries. The EA plays a central role in the security of the scheme. As in the case of regular elections, it has to be considered a trusted and secure entity. If the EA is compromised, the whole system may be defeated. For this reason, the EA should be operated off-line, by expert and trusted personnel. Ballot collecting capabilities are distributed among a set of ECs, situated at the leaves of the hierarchy. Every EC receives ballots from a relatively small group of registered voters. Every voter has to interact only with the EC which he or she is registered to. Figure 2 represents graphically the described hierarchical model. The existence of multiple concurrent ECs prevents the existence of bottlenecks in large scale elections.

The operation of our voting scheme can be divided in three phases: preliminary phase, voting phase, and shuffling phase. The procedures undertaken during each of these phases will be described in separate subsections. Specific notation will be introduced as it becomes necessary. Nonetheless, we use the following general notation:

- EA: The Election Authority.

- EC_i: Identifier of the i-th Electoral College.

- V: A particular Voter.

- P_{entity} and S_{entity}: The asymmetric key pair (respectively, public key and private key) owned by *entity*.

- $P_{entity}[M]$: Digital enveloping of message M to recipient *entity*. This comprises the symmetric encryption of M with a random session key, and the asymmetric encryption of that key with public key P_{entity}.

- $S_{entity}[M]$: The digital signature of message M created with the private key of *entity* over a digest of M. For our purposes, $S_{entity}[M]$ denotes both message M in clear and the associated signature.

- $H\{M\}$: Digest of message M produced by an one-way hash function.

- $M_1 \mid M_2$: Concatenation of messages M_1 and M_2.

- *vote*: A data string which uniquely identifies one of the voting options.

- *elect*: A data string identifying the current election (e.g. the date).

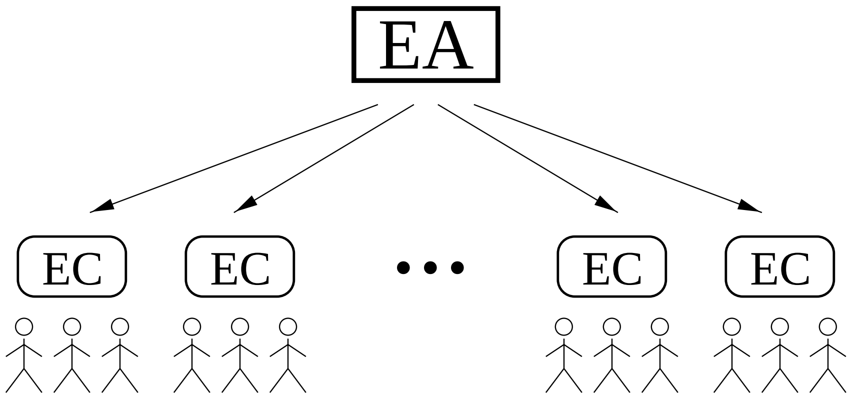

Figure 2: **Hierarchical arrangement of voting authorities.**

3.1. Preliminary phase

Prior to the preliminary phase, voters and ECs have to generate their own pairs of asymmetric keys. The public component of every key pair has to be certified by a suitable Certification Authority. Actually, the process of generation and certification of asymmetric keys would be common to other applications of the communicating society (e.g. electronic commerce or contract signing).

The first step to be undertaken during the preliminary phase consists of the creation of the electoral roll by the EA, as the list of eligible voters. ITU-T X.500 Directory Service [9] may be used to support the electoral roll in a distributed, standardized, reliable, and secure manner. It is the task of the EA to decide how many ECs will be operated, and which EC each voter will be registered to. All this information is included in the electoral roll, according to the object entries' definition given in [13].

In addition of the regular asymmetric keys generated and owned by every participant, the EA generates, during the preliminary phase, a set of asymmetric key pairs which will be useful only for the current election. For each EC, an special list of n asymmetric key pairs is generated. The public keys will be used by voters for anonymity purposes. The parameter n represents the number of times that ballot boxes will be shuffled during the shuffling phase. Since it has great impact on the security and complexity of the scheme (to be discussed in Section 4) the value of parameter n has to be accurately determined by the EA. Higher values of n mean higher security. However, lower values mean less complexity.

We denote by (P_i^1,S_i^1), (P_i^2,S_i^2), $\cdots$ (P_i^n,S_i^n) the n asymmetric key pairs (respectively, public and private key) generated by the EA for EC_i. Private keys are kept secret by the EA. The list of public keys (the order of keys is relevant) is signed by the EA, together with the identifier of the current election (*elect*) and the identifier of EC_i itself (the Distinguished Name included in the certificate of EC_i can be used as its identifier):

$$S_{EA}[P_i^1 \mid P_i^2 \mid \cdots \mid P_i^n \mid elect \mid EC_i]$$

This data is sent to EC_i together with other security-related information that we will introduce in Subsection 3.3 and with any other administrative information needed during the preliminary phase. Therefore, only one communication step is required during this phase between the EA and each EC.

3.2. Voting phase

During the voting phase, every voter contacts his or her corresponding EC to cast the desired ballot in a single session. The voting protocol executed during these voting sessions is outlined in Figure 3.

The initial step consists of the establishment of a security context between voter and electoral college, which may be based on the Simple Public-Key Mechanism (SPKM) [1] accessed through GSS-API [10]. This GSS-API mechanism consists of an initial (bilateral, for our purposes) authentication and an authenticated key establishment, which allow for further data interchanges using message authenticity, integrity and confidentiality services. Even though not all these security services would really be needed in all subsequent steps of the voting protocol, the associated overheads are low and the complexity of the implementation remains the same. Therefore, for simplicity, we have considered their use in all the steps. However, to improve readability, we do not represent the provision of these services

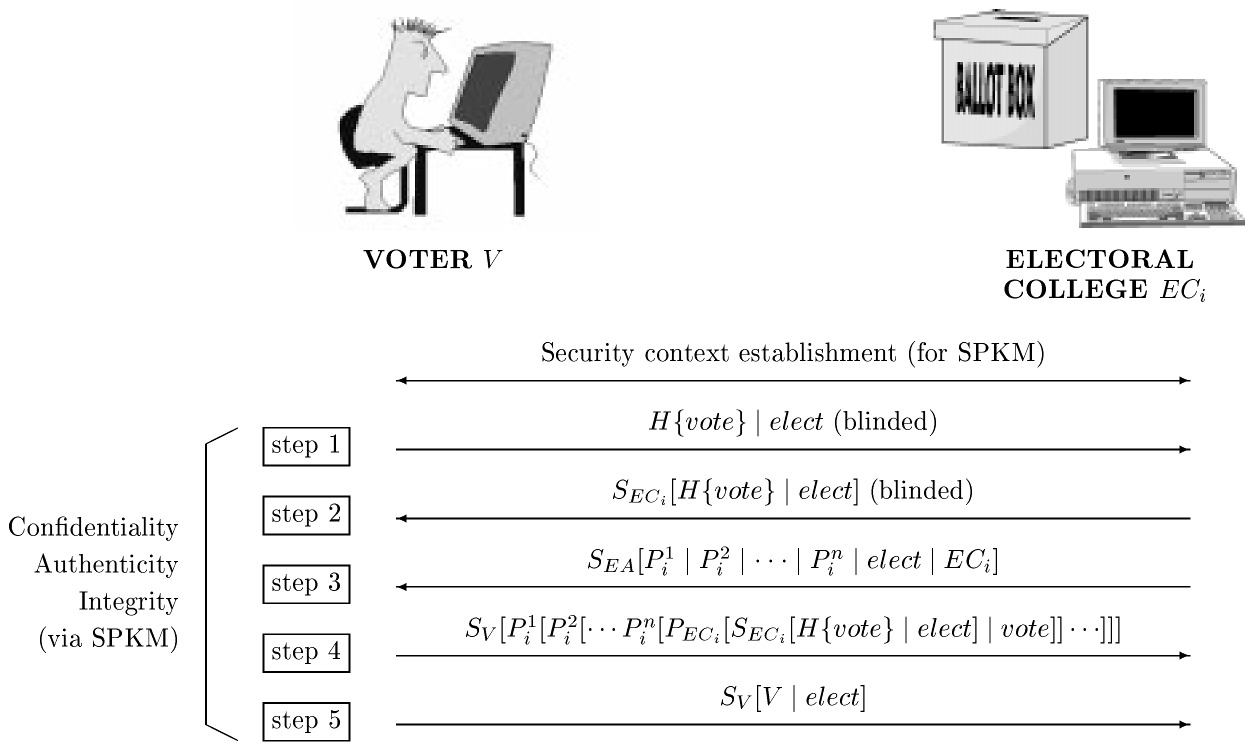
BALLOT BOX
VOTER V
ELECTORAL COLLEGE EC_i
Security context establishment (for SPKM)
step 1
$H\{vote\} \mid elect$ (blinded)
step 2
$S_{EC_i}[H\{vote\} \mid elect]$ (blinded)
step 3
$S_{EA}[P_i^1 \mid P_i^2 \mid \cdots \mid P_i^n \mid elect \mid EC_i]$
step 4
$S_V[P_i^1[P_i^2[\cdots P_i^n[P_{EC_i}[S_{EC_i}[H\{vote\} \mid elect] \mid vote]]\cdots]]]$
step 5
$S_V[V \mid elect]$
Confidentiality
Authenticity
Integrity
(via SPKM)

Figure 3: **Outline of the voting protocol.**

together with the interchanged data in each of the steps of Figure 3, but off to side. For the same reason, the protocol is presented as if it comprised five steps. Actually, the protocol would consist of only three steps, since steps 2 and 3 (and 4 and 5) would really be a single step.

As a consequence of the initial bilateral authentication, EC_i gains assurance of the identity of the voter V who is contacting it, and vice-versa. The electoral roll has to be consulted by EC_i at this point to check whether V is an eligible voter, whether V is registered to EC_i, and whether V has not voted yet. After V has been authenticated and following the consultation of the electoral roll by EC_i, V can eventually be authorized to cast a ballot. If the chosen *vote* was sent to EC_i protected by means of the confidentiality service offered by GSS-API, this would assure privacy of ballots against eavesdroppers. However, EC_i would know in which way any of its registered voters has voted. Moreover, EC_i would be able to do selective receipt: after examining the received *vote* in clear, EC_i could decide whether to accept the ballot or reject it.

To properly assure anonymity and to prevent selective receipt, EC_i has to be committed to *vote*, without knowing what *vote* really is. This could be achieved by requiring the voter to send a digest of the vote obtained through a one-way hash function, $H\{vote\}$, and then requiring EC_i to sign the received digest. However, a blind signature mechanism [4] has to be used, in order to prevent EC_i from being able to defeat anonymity, since *vote* will finally be published on the tally. A blind signature mechanism consists of masking the message to be signed through a random “blinding factor” so that the signer cannot derive any useful information from the “blinded” message. After the blinded message has been signed, the originator of the message is able to remove the blinding factor, thus obtaining the true signature of the signer over the original intended message.

In step one of the voting protocol, V sends (blinded) $H\{vote\} \mid elect$ to EC_i. Actually, if *vote* consists of few bits (less than the number of bits of the outcome of the hash function), some random padding should be added before computing the digest $H\{vote\}$. The inclusion of the string *elect* in the message sent to EC_i is justified to produce validated ballots which are valid only for the current election. In step two, V receives the blind signature of EC_i over this data. This validation string received from EC_i proves that *vote* is a valid vote for the current election and therefore it makes impossible for EC_i to later reject the ballot.

Since EC_i is blindly signing the received data, there is no way to prevent a malicious voter from accompanying $H\{vote\}$ with a wrong election's identifier. In fact, EC_i could be required to sign any other kind of more dangerous statements. To prevent these attacks, blind signatures have to be combined with cut-and-choose techniques [16]. In this way, in step one V sends in fact to EC_i a certain number, say p, of blinded messages. EC_i chooses at random one of the received messages and requests V to reveal the blinding factor of all other messages. By checking that all $p-1$ unblinded messages are of the correct form, EC_i is convinced that the message that still remains blinded is also of the same form. The probability of the voter successfully getting the signature of EC_i on a malicious message is $1/p$, which can be made as small as desired. Note that use of cut-and-choose techniques does not reveal the way in which the voter is voting since only digests of the vote with some random padding become known.

Step three of the voting protocol consists of sending to V the signed list of public keys received by EC_i from the EA during the preliminary phase. V verifies EA's signature to check the authenticity of this data. After that, V is convinced that this is the precise list of public keys the EA has generated for EC_i, for the current election. V creates now his or her ballot as the concatenation of the validation string obtained from EC_i in step two, and the desired *vote*. This ballot is enveloped to be readable only by EC_i. Next, the list of public keys P_i^j are used in turn, from P_i^n to P_i^1, to successively envelope the ballot n more times. At each enveloping operation, a new random symmetric key is chosen. This key is used to encipher all data resulting from previous enveloping. The symmetric key itself is then asymmetrically encrypted using the public key corresponding to the current step.

Figure 4 depicts the format of a successively enveloped ballot of this form. Ellipses represent asymmetric encryptions while boxes represent symmetric encryptions. All data contained within an ellipse is asymmetrically encrypted with the public key situated at the top right corner of the ellipse. In the same way, the contents of a box are symmetrically encrypted with the key situated at the top right corner of the box. All symmetric keys (in the figure denoted as K_1, K_2, $\cdots$ K_{n+1}) are random session keys. In contrast, public asymmetric keys are those already defined in Subsection 3.1.

After all enveloping steps are done, V signs the resulting data. The enveloped ballot and the associated signature are sent to EC_i in step four. EC_i adds the received ballot to its ballot box. V's signature of the ballot is stored separately. Finally, in step five, V sends to EC_i a proof that he or she has voted for this election. This proof consists of V's signature of his or her own identity and the identifier of the current election. EC_i has to store the list of all such proofs. These data will be published, together with the tally, at the end of the election.

After this last step, EC_i updates the electoral roll, crossing off V's entry to indicate that he or she has already voted. After that, the session between V and EC_i is closed.

3.3. Shuffling phase

At the end of the voting phase, when ECs stop accepting ballots from voters, each EC owns a ballot box containing successively enveloped ballots. Only the EA has knowledge of private keys needed to open the digital envelopes, except the innermost one which can be opened only by the adequate EC. The task of opening envelopes over ballots will be done by a set of mobile agents launched by the EA and that will use ECs themselves as agent servers. In this way, election tasks remain naturally distributed at the leaves of the voting hierarchy and the system keeps on being free of bottlenecks. The EA itself does not have to process even a single ballot. Even though the same goals could be achieved without using mobile agents, distribution of shuffling software and cryptographic keys can be made in a secure and elegant way using this technology. In addition, our model based on mobile agents allows for an easier coordination by the EA, while keeping the responsibilities of ECs at a minimum.

Since the agent server which hosts a mobile agent has full control over it, mobile agents should not open ballots' envelopes at the same EC which has collected the ballots. This would allow every EC to break anonymity. Instead, other ECs have to be used as agent servers. In order to simplify the reading of the text, from now on we will refer to ECs as *shuffling servers* when they act as agent servers hosting mobile agents which shuffle ballot boxes of other ECs during the shuffling phase.

At the beginning of the shuffling phase, the EA launches a mobile agent to every EC. These mobile agents consist of two parts. First, the code, which is identical for all of them. Second, the so-called baggage, data carried by mobile agents across the network and processed at agent servers. The baggage carried by every mobile agent is unique and it is initially set by the EA, before launching the mobile agent. In addition of this initial baggage, each mobile agent is responsible for fetching the ballot box from the EC where the mobile agent has been launched to. The ballot box is then

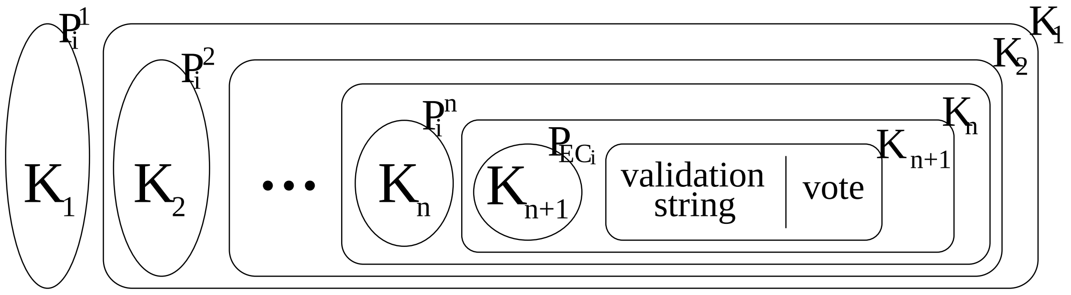

Figure 4: Format of a successively enveloped ballot.

added to the baggage and taken to n shuffling servers, in turn. We denote by EC_i^1, EC_i^2, $\cdots$ EC_i^n the shuffling servers which host the mobile agent responsible for the ballot box of EC_i. The list of shuffling servers will be referred as a *shuffling sequence*. At each visited shuffling server, a mobile agent opens one enveloping layer of ballots and shuffles the whole ballot box, thus implementing the mix functionality. The reason of performing more than a single shuffling step is to strengthen security, in the same way that a mix-net is made up of several mixes.

To be able to open digital envelopes over ballots, every mobile agent has to carry the list of required n private keys. However, private keys cannot be carried in clear. This would allow any shuffling server (and actually anyone who intercepts a mobile agent during one of its hops across the network) to gain knowledge of the private keys. With this knowledge and the collaboration of the EC which collected the ballots, anonymity may be defeated.

Private keys required to open envelopes over ballots should ideally be known only by the corresponding mobile agent, acting on behalf of the EA. Unfortunately, agent servers cannot be prevented from gaining knowledge of the data which is handled by the mobile agents they host. For this reason, the best way to protect private keys carried by mobile agents is to encrypt them, allowing decryption of each private key by only one shuffling server. Each private key has then to be used only at the server where it can be decrypted. To allow this process, the initial baggage of any mobile agent when it is launched consists of $n+1$ entries, as represented in Figure 5 (note that the first entry is a special one which contains no private key). Each entry is enveloped by the EA to be readable only by one of the shuffling servers (first entry is readable only by the EC which has collected the ballots). For coherency, in Figure 5 we have represented digital enveloping of the $n+1$ baggage entries using the same graphical notation as in Figure 4. However, symmetric keys K, K_1, $\cdots$ K_j, $\cdots$ K_n are random session keys which have no relation with those used in Figure 4. Respective entries can be identified by shuffling servers through the entries' identifiers, randomly chosen by the EA during the preliminary phase. The entries' random identifiers are represented in the figure by square boxes filled with different patterns. First entry is not accompanied by any random identifier. This entry can be identified by EC_i because of its position.

In a fair distribution of shuffling responsibilities, all ECs have to act as shuffling servers for the same number of ballot boxes. This leads to the conclusion that every EC has to be included as shuffling server in n different shuffling sequences. In this way, every server has to be provided by the EA with the n distinct baggage entries' identifiers that it will need. This is actually done during the preliminary phase, in the single communication step between the EA and every EC. In the shuffling phase, when a shuffling server receives a mobile agent, it just has to check the identifiers of the baggage entries. When a matching is found with one of the identifiers received during the preliminary phase, it means that the right baggage entry has been identified. Such entry can therefore be read only if the shuffling server's own private key is used to open the digital envelope that protects it.

Together with the private key S_i^j required to open one enveloping layer of ballots, the baggage entry readable by the shuffling server EC_i^j contains also the identity of the next server in the shuffling sequence, EC_i^{j+1}. In this way, adequate transmission of the mobile agent is allowed. The identity of the first shuffling server, EC_i^1, is included in the first entry, readable only by EC_i.

When the mobile agent sent by the EA to EC_i is forwarded by EC_i to EC_i^1, its baggage comprises two parts. First, the enveloped entries shown in Figure 5 which are set initially by the EA, and that will re-

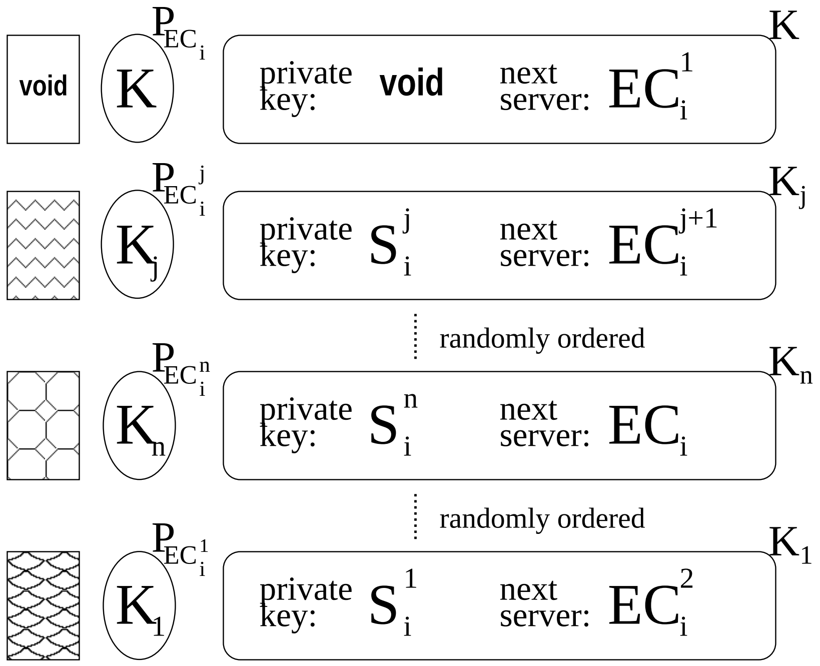

Figure 5: **Baggage carried by a mobile agent when launched.**

main static along the whole shuffling sequence. Second, the ballot box fetched from EC_i which will be processed and modified at each shuffling server. When the mobile agent is received by a shuffling server, the mobile agent is first provided by the server with the adequate private key once the corresponding baggage entry has been identified and read. With this private key, the mobile agent opens the outermost digital envelope that protects all ballots in the ballot box. After this is done, the contents of ballots still appear as garbled bits, due to next enveloping layers. However, the fact of removing one digital envelope has shortened ballots. To overcome this, the mobile agent has to append random padding at the end of each ballot, following the scheme shown in Figure 6. The number of bits added must equal the number of bits removed, that is the length of the asymmetric encryption of a symmetric key. To assure that adding random bits at the end of a ballot does not affect the vote itself, the encryption algorithm should be such that correct decryption of a given block depends only on previous blocks, but not on following blocks. This condition is fulfilled by most encryption algorithms. After this step of opening envelope and padding has been done for all ballots, the mobile agent shuffles the ballot box (i.e. it randomly exchanges the position of all ballots). Next, the mobile agent, with its new version of ballot box, has to be sent to the next shuffling server in the sequence.

At the end of the shuffling sequence, shuffling server EC_i^n obtains the identity of the original EC_i as the next shuffling server where to send the mobile agent (see Figure 5). However, after inspecting the static part of the baggage of the received mobile agent, EC_i detects that in fact the mobile agent is the same that originally fetched its ballot box. As a conclusion, the received ballot box is its own one which has already been shuffled n times. Ballots are already protected only by the innermost envelope. Therefore, EC_i can use its own public key to open the last envelope over ballots, obtaining finally a list of readable ballots. Every ballot consists of a validation string produced by EC_i itself during the voting phase, followed by the string *vote*, and altogether followed by unreadable garbled padding. Since validation strings were produced through blind signatures, they cannot be used to link ballots with voters. Still, these validation strings allow EC_i to verify that ballots have not been tampered with during the shuffling process.

4. Discussion

A malicious or tampered mobile agent could accumulate or reveal knowledge of all private keys as they are being used along the shuffling sequence. This could lead to defeat of anonymity. For this reason, whenever a mobile agent is received by a shuffling server, the

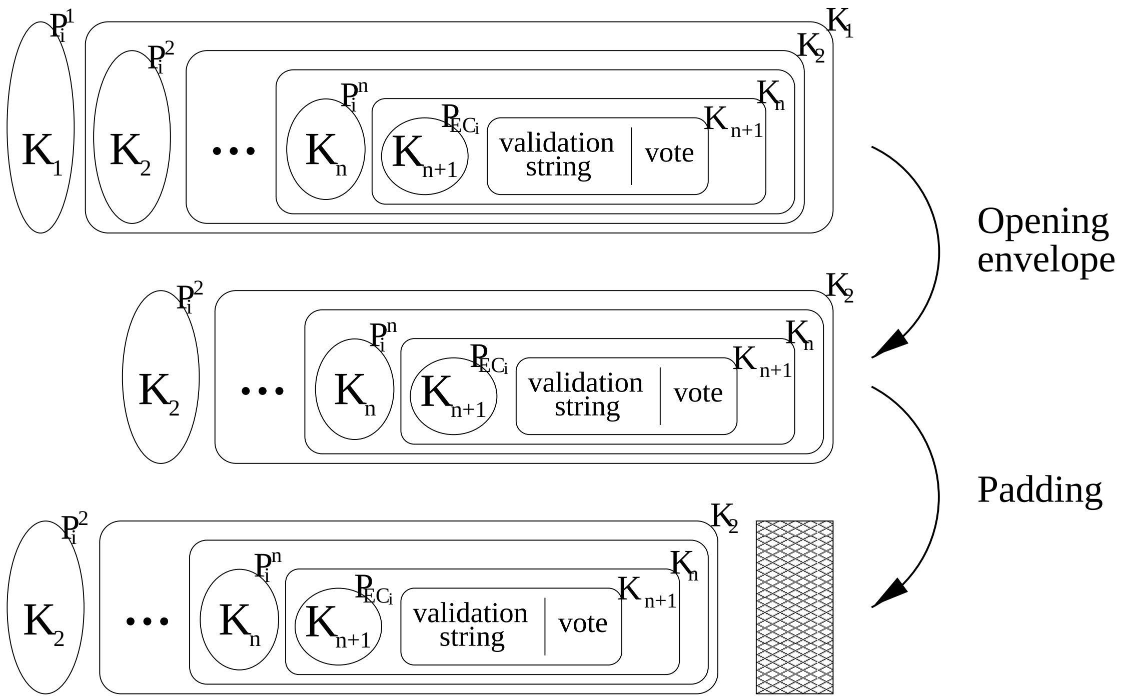

Figure 6: **Opening the outermost envelope over a ballot.**

authenticity and integrity of the mobile agent and its initial baggage must be verified by using the security infrastructure described in [5]. Basically, the EA's signature of each mobile agent when it is launched is used for this purpose. Since the carried ballot box is not initially signed by the EA and it actually changes from server to server, its authenticity and integrity have to be protected at each hop by means of a digital signature applied by the shuffling servers themselves. Any time a server forwards a mobile agent, it must first sign the shuffled ballot box. This signature can be verified by the next server. After the signature has been verified, a copy of the received ballot box is stored (the objective will become clear in Subsection 4.2).

The format of ballots in our scheme is very similar to the one when using a mix-net composed of n mixes as defined in [8]. In both cases, the size of ballots corresponds to the size of the vote portion plus n times the size of the asymmetric encryption of a symmetric key. Ballots are transmitted across the network $n + 1$ times, in both cases again. However, when a mix-net is used, every ballot is transmitted independently and therefore it requires its own datagram headers. On the other hand, our scheme requires transmission of the code of the mobile agents themselves, together with ballot boxes. Nonetheless, under the parameters expected for regular elections, the size of the mobile agents' code would not be significant. As a conclusion, our scheme implies roughly the same communication costs than a mix-net composed of n mixes.

Not only communication costs, but also the security level of our voting scheme is equivalent to the use of a mix-net. Anonymity of ballots is guaranteed as long as at least one out of n entities remains honest. As a consequence, the system becomes more secure as n is increased. Unfortunately, to increase n means also to increase network's load during the shuffling phase. Therefore, the adequate balance between security and communication costs must be found by the EA when deciding the value of n.

In the following discussion we will assume the worst scenario, i.e. when only one shuffling server in a shuffling sequence is honest, while all other $n - 1$ servers collude with the original EC to try to defeat anonymity of ballots. If collusion takes place, they cannot skip the single honest shuffling server (independently of its position within the shuffling sequence). The corresponding enveloping layer of ballots can be opened only by using a private key, which is, in turn, enveloped to be readable only by the honest server. Therefore, the only way for a collusion of $n - 1$ shuffling servers to try to defeat anonymity is by analyzing the input and the output to/from the single honest server. Both passive and active attacks are possible.

4.1. Passive attacks

Passive attacks against the single honest shuffling server consist of trying to correlate the incoming ballots delivered to it, with the resulting outgoing shuffled ballots. This correlation could be done either by contents or by length.

Correlation by contents is not possible because the enveloping layer of ballots that is opened at the honest server involves random session keys chosen by voters. These symmetric keys are known only by the mobile agent when it is executed at the honest shuffling server. The mobile agent destroys (actively erases from memory) these symmetric keys after being used. In this way, any passive attacker would have to guess the key used in each encryption, in order to be able to encrypt again outgoing ballots in the same way they were encrypted when reaching the honest server. This requires trying all possible keys, i.e. a brute-force attack. The padding that is appended to ballots at the honest server provides no information at all, since it is random. Correlation by length is also prevented because all ballots are always of the same length.

Note that all ballots in a ballot box arrive to (and leave from) the honest shuffling server at the same time. As a consequence, passive attacks by causality (i.e. time of arrival and departure) are not possible. This represents a remarkable difference from the use of a mix-net, since in that case ballots arrive to the mixes when they are cast by voters and hence at different times.

4.2. Active attacks

Ballots collected during the voting phase by different ECs are enveloped using different lists of public keys. This makes attacks based on exchanging enveloped ballots between ECs unfeasible, because the exchanged ballots would not be successfully decrypted in other shuffling sequences. For the same reason, it is not possible to include in a ballot box some ballots from previous elections. Replay attacks based on sending the same ballot box (or some variations of it) more than one time are prevented if shuffling servers record the already-used baggage entry identifiers. These identifiers cannot be modified by attackers because of the EA's signature of the initial baggage of each mobile agent. Therefore, active attacks are restricted to those based on modifying in some way a ballot box before sending it to the honest shuffling server. The objective of such modifications is to allow identification of some ballots after the shuffling performed at the honest server. At the end of the shuffling sequence, identified ballots could then be traced back to the voters who cast them.

One way to modify a ballot box in order to allow future identification of a ballot consists of duplicating the ballot inside the ballot box. Nonetheless, this attack can be easily detected by the mobile agent when it is executed at the honest shuffling server, by checking that all ballots are different. This can be done by comparing the symmetric keys used in the ballots' outermost envelopes. If two symmetric keys turn out to be equal, then with very high probability a ballot has been duplicated (the chances of two random symmetric keys being accidentally equal are insignificant). Attackers could duplicate a ballot and then modify the (asymmetrically encrypted) symmetric key in the duplicate. However, if only a single bit was changed, that would make the duplicate absolutely indecipherable at the end of the shuffling sequence.

Another possible active attack consists of isolating a ballot inside the ballot box. To do this, all ballots except one are removed. The ballot box can then be refilled again with other decoy ballots. Decoy ballots can even end up looking like valid ballots if the original EC (where the ballot box was fetched from) participates in the attack. Although anonymity of a single voter would be affected, this kind of attack represents a powerful threat in the sense that it cannot be detected by the honest shuffling server. Moreover, the only possible countermeasures increase both the complexity of the scheme and the network's load and, actually, the attack is still not thwarted completely. For these reasons, our approach does not try to counteract the attack when it takes place. Instead, a-posteriori measures are taken. First, note that the attack becomes very apparent, at the end of the shuffling phase. Only one voter of those registered to a certain EC would be satisfied with the published tally. Since our voting scheme is verifiable, the rest of voters would detect that their respective ballots have not been counted. In such case, the tally is suspended by the EA, and the causes of the problem are analyzed. Even though the attack is very apparent, the identity of the attacker is not obvious. It could have been the EC, trying to defeat anonymity of a voter. It could also happen that one (or some) of the shuffling servers eliminated ballots. It is even possible that some voters conspired, sending invalid ballots during the voting phase to the EC, in an attempt to undermine the EC's trustworthiness or the reliability of the whole voting scheme.

To find out the origin of the attack, the EA asks every server of the shuffling sequence for a copy of the ballot box received from the previous server. Authenticity and integrity of these instances of ballot boxes can be verified by means of digital signatures performed by shuffling servers before they forward the mobile agent. Since the EA knows all private keys used by the mobile agent along the shuffling sequence, it is able to repeat locally all steps, comparing the outcomes with the signed ballot boxes with which it has been provided. If one of the shuffling servers has removed ballots from the ballot box, the shuffling server is identified. If this is not the case, then the root of the problem is located at the original ballot box fetched by the mobile agent from the original EC. This means that either the EC has deliberately substituted certain validated ballots by invalid ballots, or that some voters have sent invalid ballots to the EC. Since any EC stores the signatures of voters of the original enveloped ballots, the EA can easily discern if the attacker is the EC or, alternatively, some voters. As a conclusion, given the ease of detection and fear of reprisals, it can be assumed that this kind of attack would not take place.

Finally, active attacks may consist of the modification of a ballot itself. If this modification affects any of the symmetric keys used in the digital envelopes, it would prevent proper decryption of the ballot. As a consequence, anonymity of the affected voter would not be defeated. In the same way, assuming that the encryption algorithm is used in a mode such that correct decryption of a given block depends on previous blocks, modification of even a single bit of the encrypted blocks containing the vote portion of the ballot makes the string *vote* indecipherable since it is situated in the last block. Attackers can modify a ballot without affecting either symmetric keys or the vote portion, but only the random padding, e.g. by adding some concealed identification information. However, this action is absolutely useless to allow future identification of the ballot because, at the honest server, the padding will be symmetrically "decrypted" with the symmetric key used in the (at that moment) outermost digital envelope of each ballot. This will actually produce the effect of enciphering the padding with a key known only by the mobile agent.

4.3. Reducing the risk of collusion

When the tally is published, no correlation is possible between tabulated ballots and the original enveloped ballots, unless all of the servers of a shuffling sequence collude with the original EC which collected the ballots. Collusion attempts before the shuffling phase are very unlikely because ECs do not know the identities of the n particular shuffling servers that will constitute their shuffling sequence. To make collusion attempts during the shuffling phase more difficult, we require that knowledge gained by ECs about their shuffling sequences is minimal. At the beginning of the shuffling phase, when an EC receives the corresponding mobile agent from the EA, it is able to read from the agent's baggage only the identity of the first shuffling server. At the end of the shuffling sequence, when the mobile agent is received back again, the EC gains knowledge of the last server. The identities of all intermediate servers remain unknown.

Shuffling servers in the sequence are not able to know from which EC the ballot box was fetched. This is true even for the first server in the sequence, which does not know whether the mobile agent is received from the original EC or from a previous shuffling server. Baggage entries' random identifiers provide no knowledge about the identities of the corresponding shuffling servers. Therefore, any server in a shuffling sequence gains knowledge only about the identities of the previous server (where the mobile agent comes from) and the next server (read from the baggage entry), but not of the rest.

The time needed for transmission and for processing at each shuffling server may be different for every mobile agent. As a consequence, there is no synchronization between the number of shuffling servers already visited by every mobile agent. This feature can actually be reinforced if the EA launches the mobile agents at different times. Therefore, when a shuffling server receives a mobile agent, it cannot deduce from the time of arrival the number of shuffling servers previously visited by the mobile agent (i.e. its own position within the shuffling sequence). Entries in the baggage are carried in random order and therefore their positions provide also no knowledge. The carried ballots have always the same form and size independently of the number of servers previously visited. Each ballot consists of two parts. The first part is the successively enveloped vote portion, while the second part is random padding that has been successively symmetrically enciphered. As the ballot box is shuffled at new shuffling servers, the first part becomes smaller, and the second part larger, but the overall length remains constant. However, shuffling servers are not able to determine the border between the two parts because they both look as random bits.

5. Security of the election

After the discussion on how shuffling sequences provide anonymity of ballots in our voting scheme, we turn now to analyze the fulfillment of the rest of security requirements listed in Section 1.

5.1. Accuracy

Accuracy in our voting scheme is guaranteed because all validated ballots incorporate digital signatures of the corresponding ECs over the digest of *vote*. These signatures assure the authenticity and integrity of votes. Nobody other than the corresponding EC is able to modify a *vote*, while still offering a valid signature on the digest of it. Still, if a certain EC modifies (or removes) a ballot, the affected voter is able to prove the fraud to the EA. The voter just has to show the validation string obtained during step two of the voting protocol (note that the one-way hash function used to compute the digest of votes must be collision-free in order to prevent attacks from malicious voters). The EA can check that no ballot in the tally appears to match the voter's validation string. This claim can actually be made as a public objection to the tally. The voter can openly show the validation string to everyone without revealing in which way he or she voted, because the validation string includes only the digest of the vote and not the vote itself. The concept of open objections to the tally was first introduced in [15]. However, the solution offered in such reference has a practical problem since it combines generation of random asymmetric key pairs by voters with blind signatures (and therefore, from a practical point of view, with cut-and-choose techniques). As a consequence, any voter is forced to generate many pairs of asymmetric keys, what requires significant time. Use of one-way hash functions, as we have described, serves the same purpose and it is several orders of magnitude faster. Furthermore, in contrast with the proposal in [15], if a malicious voter casts purposely an invalid ballot, this can be detected by the EA in our scheme, as it has been discussed in previous section.

Addition of invalid ballots to a certain ballot box can only be done by the corresponding EC, because no other entities can create the adequate signature on validation strings. To prevent any malicious EC from adding invalid ballots on behalf of abstaining voters, the results of the election are published in our scheme using two lists, according to the format shown in Figure 7. The main list of ballots is accompanied by a secondary list containing the identities of all voters who have voted, signed by themselves. This data is collected from voters at step five of the voting protocol. If the number of entries in the list of ballots was greater than the number of voters who have voted, it would mean cheating by the involved EC.

5.2. Democracy

Democracy in our voting scheme is assured through proper authentication of voters to ECs, and adequate policies of access control to the electoral roll. Secure access to X.500 Directory ensures that only the EA is able to register eligible voters to the electoral roll, and only ECs are able to set the boolean attribute which indicates that one of the registered voters has already voted. Not only voters are prevented from attacking the democracy requirement, but also ECs cannot do it without being detected. If a certain EC accepts ballots from non-eligible users (or if it accepts more than one ballot from the same eligible voter), this would be reflected when the results are published. The list of ballots would contain more entries than the list of eligible voters who voted. Therefore, these cases can be treated analogously, as if the EC added ballots on its own.

5.3. Privacy

The second privacy property, uncoercibility, requires that voters are not able to prove in which way they voted. The objective is to prevent coercion. Coercion may consist either of buying of votes or of extortion (intimidation to cast a particular vote). If voters do not obtain any proof of the vote cast, then coercers cannot act with certainty of success. Uncoercibility is a very particular issue which needs some hardware components to be included into the operation of the voting scheme. For clarity, we have not considered uncoercibility in the design of our voting scheme. However, the techniques presented in [14], based on smart-cards, could be used to assure it.

The third privacy property, fairness, requires that all votes remain secret while the voting phase is not completed. The objective is to prevent anyone (normally ballot collecting authorities) from knowing intermediate results of the election. Such knowledge could be used to affect voters' behavior. In our voting scheme, fairness is assured because all ballots are stored enveloped in the respective ballot boxes. Private keys required to open the digital envelopes are known during the voting phase only by the EA. In addition, the innermost envelope can be opened only by the adequate EC. Only when all ECs are closed, the shuffling phase is started and envelopes are opened.

5.4. Verifiability

Any voter can verify that his or her ballot has been counted by just looking for it in the published voting results. If the ballot does not appear, then the affected voter can do a public objection, as it has been commented.

$\vdots$	
$S_{EC_i}[H\{vote\} \mid elect]$	$vote$
$\cdot$	

$\cdot$
$S_V[V \mid elect]$
$\vdots$

Figure 7: Format of the published tallies.

6. Conclusions

This paper describes a realistic electronic voting scheme, suitable to be implemented over large internets, which fulfills all commonly accepted security requirements, except uncoercibility. Anonymity of ballots is provided by using the same techniques which constitute the essence of mix-nets. However, the mix functionality is integrated into the design of the voting scheme itself and therefore no external mixes are needed. Mobile agents are used to implement the mix concept. They are hosted by the components of the voting scheme themselves.

Even though the security level and communication costs are the same as in the classical architecture based on a mix-net, our scheme improves some aspects. The fairness requirement is fulfilled since all ballots remain enveloped while the voting phase is not completed. The voter casts the desired ballot during a single session established with the corresponding Electoral College. If a ballot is removed from the tally, then there is a mechanism by which the Election Authority can identify the attacker. Problems of low traffic incoming at any shuffling server disappear, because shuffling is performed when all ballots are already cast. The system is totally self-contained, and therefore its implementation is simplified.

Our group is working on a 100% Java implementation of a large scale electronic voting system. The solution to anonymity currently considered in our system requires proper certification of the hardware and software which implement the ECs. Integration of the solution presented in this paper into the development of our system's prototype would allow to analyze its robustness, performances and costs.

7. Acknowledgments

Special thanks to Professor Sead Muftic and the Department of Computer and Systems Sciences of the Stockholm University, for their collaboration during the elaboration of this paper. This work has been partially funded by the Spanish Government Commission CICYT, through its grant TEL97–0663.

References

[1] Adams, C. (1996) Internet RFC **2025**: SPKM: The Simple Public-Key GSS-API Mechanism. *ftp://ftp.isi.edu/in-notes/rfc2025.txt.*

[2] Cranor, L.F. and Cytron, R.K. (1996) Design and Implementation of a Practical Security-Conscious Electronic Polling System. *Washington University Report WUCS-96-02. http://www.ccrc.wustl.edu/~ lorracks/sensus/.*

[3] Chaum, D. (1981) Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms. *Communications of the ACM*, **24**, 84–88.

[4] Chaum, D. (1983) Blind Signatures for Untraceable Payments, in *Advances in Cryptology – CRYPTO '82*, (Plenum Press), 199–203.

[5] Cheng, Y. (1997) A Comprehensive Security Infrastructure for Mobile Agents. *Licentium Thesis, Stockholm University/KTH.*

[6] Cottrell, L. (1998) Mixmaster and Remailer Attacks. *http://obscura.com/~ loki/remailer-essay.html.*

[7] Fujioka, A. and Okamoto, T. and Ohta, K. (1992) A Practical Secret Voting Scheme for Large Scale Elections, in *AUSCRYPT '92*, LNCS **718** (Springer-Verlag), 244–251.

[8] Gülcü, C. and Tsudik, G. (1996) Mixing Email with BABEL, in *Symposium on Network and Distributed Systems Security (NDSS '96)*, Proceedings in CD-ROM ISBN 0-8186-9154-9 (Internet Society), 15 pages.

[9] ITU-T (1993) Recommendation X.500 (11/93) – Information technology – Open Systems Interconnection – The directory: Overview of Concepts, Models and Services.

[10] Linn, J. (1993) Internet RFC **1508**: Generic Security Service Application Program Interface. *ftp://ftp.isi.edu/in-notes/rfc1508.txt*.

[11] Pfitzmann, B. (1994) Breaking an Efficient Anonymous Channel, in *EUROCRYPT '94*, LNCS **950** (Springer-Verlag), 339–348.

[12] Park, C. and Itoh, K. and Kurosawa, K. (1993) Efficient Anonymous Channel and All/Nothing Election Scheme, in *EUROCRYPT '93*, LNCS **765** (Springer-Verlag), 248–259.

[13] Riera, A. and Borrell, J. and Rifà, J. (1997) Large Scale Elections by Coordinating Electoral Colleges, in *IFIP SEC '97, Information Security in Research and Business* (Chapman&Hall), 349–362.

[14] Riera, A. and Borrell, J. and Rifà, J. (1998) An Uncoercible Verifiable Electronic Voting Protocol, in *IFIP SEC '98, IT Global Security* (Austrian Computer Society), 206–215.

[15] Sako, K. (1994) Electronic Voting Scheme Allowing Open Objection to the Tally. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences Vol.E77-A*, **1**, 24–30.

[16] Schneier, B. (1996) Applied Cryptography. Protocols, Algorithms and Source Code in C. 2nd ed. *John Wiley & Sons.*