ErgoGroup

E-vote 2011

System Architecture

eVote

V 1.3

Change history

Date	Version	Description	Author
11.11.2010	0.1	Initial	SWIN
02.12.2010	0.2	Update	DALC
09.12.2010	1.0	First published version	SWIN
02.06.2011	1.1	Reviewed	DALC
03.06.2011	1.2	Added disclaimer, and copyright and next version target	JB
13.06.2011	1.3	Return codes and key management sections updated	DALC

DISCLAIMER: This document is a work in progress and some information in it might be obsolete, inaccurate or might be missing. Regular updates will be made to the document. This disclaimer will be also updated to reflect the state of the document.

NEXT VERSION TARGET DATE:

Version	Date	Author	Comments
1.4	27.06.2011	Scytl	

Approval

Approved by	Role	Sign	Date
Svein Endresen	Project Manager		22.03.2011
Svein Winje	Technical Architect		22.03.2011
Dan Sørensen	Customer		

Contents

1. Electronic Voting (eVoting)

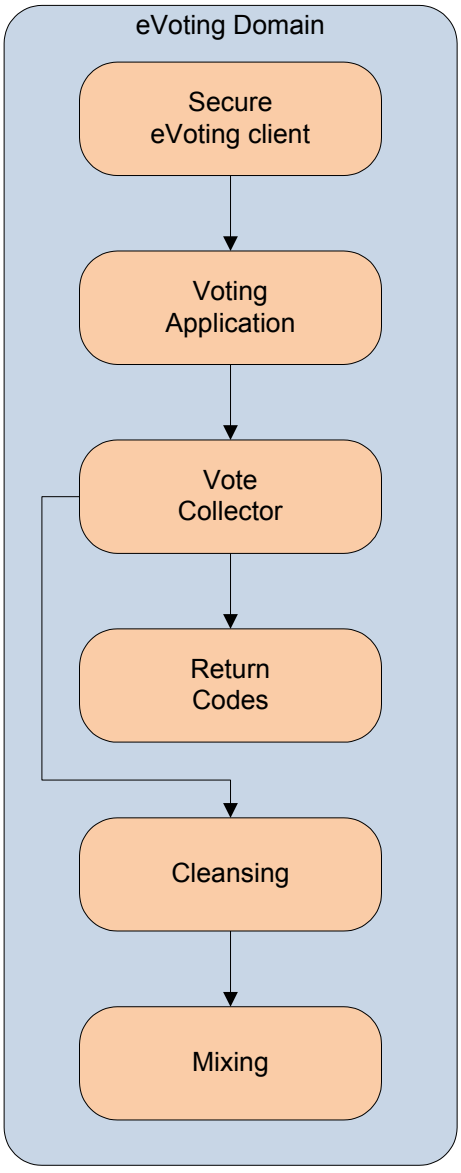
1.1 Architecture goal and scope

The Electronic Voting (eVoting) architecture guarantees that service to voters will be required fulfilling both functional and non functional application requirements, focusing mostly in the security and privacy of the voter, and the performance of the application in the second place.

1.2 Essential use cases

Electronic Voting (eVoting) will cover all user stories related to use case 2.1 as described in the contract.
Use case 2.1 is implemented using a set of servers and a voting client that allow establishing a secure channel from the voter to the Electoral Board by means of secure voting protocol.

1.3 Logical flow



Secure Voting Client
A Java applet that runs on the voting computer and encrypts the vote on the client side, to guarantee voter privacy, even from insider attacks.

Voting Application
This component contains an accessible web application that handles all the interaction with the voter. It presents the user interface, downloads the Secure Voting Client applet and acts as a proxy to the Vote Collector and Return Codes.

Vote Collector Server
This component is a web service that stores the votes securely.

Return codes Server
This component receives the votes from the vote collector and sends return codes for the voter to verify his vote. The return codes are sent using an external SMS Gateway.

Cleansing Server
This component pre processes the encrypted votes before being sent to the Mixing, eliminating duplicates and verifying against the electoral roll. It produces an EML with the encrypted votes to be sent to the Mixing.

Mixing Server
Breaks the correlation between the voter and vote and it decrypts the vote afterwards. It produces an EML with the decrypted vote to be sent to the Settlement.

1.4 Interfaces between subsystems

The eVoting system has the following interfaces.

Receive from administrative system:

- Election configuration and language translations in EML
- File containing the Electoral Roll
- Voter eliminations during advance voting (to be removed during cleansing)

Publish to administrative system:

- Counts of votes

Web service:

- Access to Electoral Roll through web service for voter authorization

MinID:

- Access to service MinID for voter authentication

1.4.1 Return Codes and Voting Card Generation

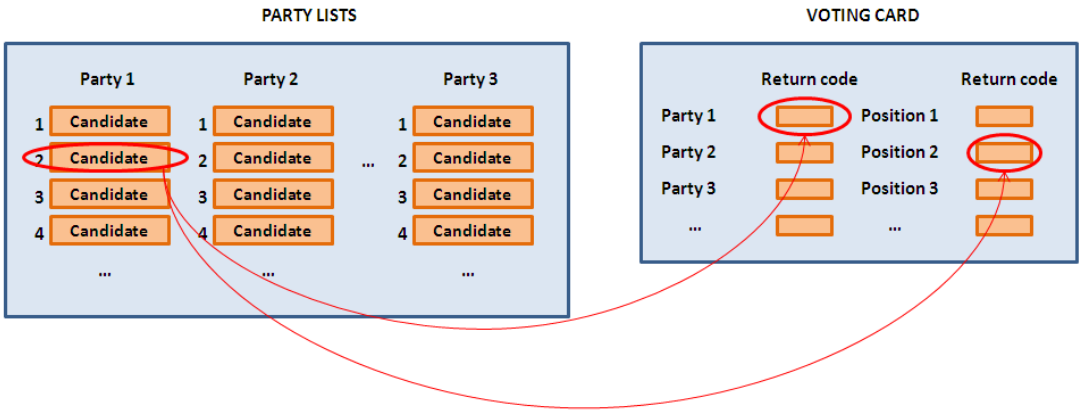
The Voting Card is a paper sheet containing a unique Return Code for each party list and for each position in the party list.

Voting Cards are used for the verification of the correct recording of the voter's intent (cast as intended) by the VCS. To this end, after the voter has cast a vote, the RCG returns the Return Codes of the parties selected by the voter, and the Return Codes corresponding to the positions of the selected candidates in the party lists. Therefore, voters can verify if the Return Codes returned by the RCG match with the ones available on the Voting Card for the same selected voting options. Since the Voting Card is only available on paper, it is impossible for an attacker controlling the voter terminal to know in advance the values of the selected voting options.

The Voting Card will contain two sets of Return Codes:

- Party Codes
- Position Codes

Using this configuration, the voter will be able to verify, the party/parties she has voted for and the candidates she has selected looking to their positions in the party list.

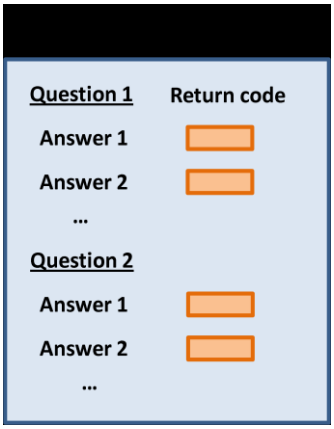


The presentation of the candidates' Return Codes in Position Codes is aimed to achieve a high degree of usability and understandability when the voter receives these codes through SMS. The Return Code generation in the voting phase is divided in three steps:

- Calculation of Return Codes representing the parties.
- Calculation of Return Codes representing the candidates (the same process than with the parties).
- Mapping of these Return Codes to Position Codes, thanks to a set of tables stored in a database in the RCG.

- Mapping of the party Return Codes to PartyCodes, thanks to the same set of tables in the RCG: the Return Codes need to be translated into shortened codes (in this case, PartyCodes) to be sent to the voters, and duplication controls need to be applied to them.
- Therefore, there are two types of party return codes:
    - The PartyCodes are the values that will be printed in the Voting Cards (shortened codes with duplication control).
    - The Party Return Codes are the values that are calculated during the voting process. They are mapped to the PartyCodes using tables stored in the RCG's database.

In case of referendums, with several possible answers for each question, we have a Voting Card like this:



The set of Return Codes in the voting card consists in all the possible answers, and they will be generated in the same way that the PartyCodes in municipal elections:

The AnswerCodes are generated in the following way:

- Calculation of Answer Return Codes representing the answers.
- Mapping of the Answer Return Codes to AnswerCodes, thanks to the set of tables in the RCG: the Answer Return Codes need to be translated into shortened codes (in this case, AnswerCodes) to be sent to the voters, and duplication controls need to be applied to them.

Therefore, there are two types of answer return codes:

- The AnswerCodes are the values that will be printed in the Voting Cards (shortened codes with duplication control).
- The Answer Return Codes are the values that are calculated during the voting process. They are mapped to the AnswerCodes using tables stored in the RCG's database.

After that mapping, an example of SMS sent to the voter could be:
"The Voting Service has registered your vote for party list XXXX, and candidates in positions XXXX and XXXX from that party. Additionally, from the party list XXXX, the candidates in positions XXXX and XXXX have been marked…"

1.4.1.1 Return Code Generation in the Election Configuration Phase

Since the **Position Codes** are equal for all the parties and different for each voter, they can be calculated as:

- $PositionCode_i = HMAC(ID_n||position, K_{RCG})$ ← take last four decimal numbers.

The same for the **PartyCodes** to be printed in the Voting Card. They are calculated as:

- $PartyCode_i = HMAC(ID_n||party\_num, K_{RCG})$ ← take last four decimal numbers.

The party_num values is a number assigned to the party during the configuration phase (can be consecutive).

Although they are not present in the Voting Card, the **Candidate Codes** and also the **Party Return Codes** and **Answer Return Codes** will be generated at the Election Configuration phase in order to configure the RCG's database. This generation is based on the following components:

- Secret parameter si: a unique 2048-bit code for each voter.
- Function *d*: an HMAC function used with the symmetric key of the RCG, K_{RCG}.

The **Party Return Codes** (different for each party and voter) are calculated from the choices selected by the voter, using the parameter s_i and the function *d*:

- $s_i=K_{VCS}^{IDn}$ mod p, where ID_n is the voter identifier in the election and p is the ElGamal public module.
- $P_i'=P_i^{si}$ mod p.
- $PartyReturnCode_i = HMAC(P_i'||ID_n, K_{RCG})$

The **Answer Return Codes** (different for each answer and voter) are calculated from the choices selected by the voter, using the parameter s_i and the function *d* (the same as in Party Return Codes):

- $s_i=K_{VCS}^{IDn}$ mod p, where ID_n is the voter identifier in the election and p is the ElGamal public modulo.
- $A_i'=A_i^{si}$ mod p, where A_i is the prime number representing the answer.
- $AnswerReturnCode_i = HMAC(A_i'||ID_n, K_{RCG})$

The **Candidate Codes** (different for each candidate and voter) are calculated similarly:

- $s_i=K_{VCS}^{IDn}$ mod p, where ID_n is the voter identifier in the election and p is the ElGamal public modulo.
- $C_i'=C_i^{si}$ mod p.
- $CandidateCode_i = HMAC(C_i'||ID_n, K_{RCG})$

Note: these return codes are not trunked to the last 4 digits.

The generation of the Return Codes in the election configuration phase will be split in two steps, and each one will be executed in a different module:

VCS

The secret parameters si are bind to the voter identity, and they are calculated from a secret key in the VCS. These secret parameters will be generated at the moment they are used (during the voting phase) with a modular exponentiation using the KVCS key (2048 bits), and the voter election identifier IDn as:

$$si=KVCSIDn \bmod p.$$

In the election configuration phase, the VCS calculates, for each parameter si, the value of gsi.
The VCS also computes a partial calculation of the Return Codes as f(v)si, where f(v) represents the mapping of the voting options (v) to prime numbers. So, we have the prime number raised to the parameter si.

This partial calculation of the Return Codes is passed to the Return Code Generator disconnected from the voter identities IDn. Therefore, the Return Code Generator will know the final values of the Return Codes, but not the identity of the voter to which they are bind to. This enforces the privacy of the contents of the Voting Cards.

The VCS will send to the RCG the partial calculation of the Return Codes bind to voting card identifiers Bid, the partial calculation of the Position Codes, and the values gsi, connected to the respective voter identities IDn.

Computation steps in VCS' (Election Configuration)

1. For each ID_n (voter identifier, or Social Security Number) compute:
$$s_i=K_{VCS}^{IDn} \bmod p.$$
2. For each ID_n compute g^{si} *mod p* and create the list A:{$ID_n \leftrightarrow g^{si}$} for all the ID_n's.
3. For each ID_n and for each voting option (parties and candidates) compute:

   $P_i'=f(v_i)^{si} \bmod p$, where $f(v_i)$ is the prime number assigned to a specific party.
   $C_i'=f(v_i)^{si} \bmod p$, where $f(v_i)$ is the prime number assigned to a specific candidate.
   $A_i'=f(v_i)^{si} \bmod p$, where $f(v_i)$ is the prime number assigned to a specific answer (for referendums).

4. For each ID_n compute B_{id}=HMAC(ID_n, K_{VCS}), **using the last 256 bits of the key K_{VCS}**, and create the list B:{$ID_n \leftrightarrow B_{id}$}.
5. Encrypt the list B with the public key of the printer service: $E_{Pprinter}(B)=B'$.
6. Create the list C:{$B_{id} \leftrightarrow \{P_i', C_i'\}$} (or C:{$B_{id} \leftrightarrow \{A_i'\}$} in case of referendums), **scramble the list** so that B_{id} values are not in the same order than in list A. The codes P_i', C_i' should be still associated to the original Party and Candidate names (in case of referendums, the same for the codes A_i' and the answer names).
7. Send A and C lists to the RCG'.
8. Send B' list to the printing service.

RCG

The RCG stores the values gsi bind to the voter identities for further verifications (verification of Zero Knowledge Proofs), and calculates the final values of the Return Codes using the partial calculation from the VCS, Pi', Ci' or Ai', a HMAC function (based on SHA256), a symmetric key KRCG (256bits), and the voter election identifier IDn:

ReturnCodei=HMAC((Pi', Ci' or Ai') || IDn, KRCG),

Computation steps in RCG' (Election Configuration) Part 1: Creation of Voting Cards

9. Receive and store the list A:{$ID_n \leftrightarrow g^{si}$} from VCS'.
10. Receive list C:{$B_{id} \leftrightarrow$ {P_i', C_i', or A_i'}} from VCS' and perform the following operations:
    a. Compute for each B_{id}, party and candidate (or answer for referendums):

       PartyReturnCodei=HMAC(Pi'|| IDn, KRCG)
       AnswerReturnCodei=HMAC(Ai'|| IDn, KRCG)
       CandidateCodei=HMAC(Ci'|| IDn, KRCG)
       The codes PartyReturnCode and CandidateCode should be still associated to the original Party and Candidate names (in case of referendums, the same for the codes Ai' and the answer names).
    b. Compute for each B_{id} an additional value that will represent the empty vote:

       EmptyVoteCode=HMAC(1||ID_n, K_{RCG}), **and truncate to the last 4 digits.**

11. For each B_{id} and for each possible position in a party list, calculate:
    a. $PositionCode_i$ = HMAC(ID_n||position, K_{RCG}), **truncate to the last 4 digits.** ← The codes PositionCode should be still associated to the original position name.
    b. $PartyCode_i$ = HMAC(ID_n||party_num, K_{RCG}), **truncate to the last 4 digits.** ← The codes PartyCode should be still associated to the original party name.
    c. $AnswerCode_i$ = HMAC(ID_n||answer_num, K_{RCG}), **truncate to the last 4 digits.** ← The codes AnswerCode should be still associated to the original answer name.

***IMPORTANT**: while generating these codes, it is very important to check that two PositionCodes, two PartyCodes, or two AnswerCodes have not the same value for the same voting card/voter, or that they have the same value than the EmptyVoteCode. In case a value is duplicated, calculate a new value by adding the num max of positions or parties to the current number to calculate the new Return Code. Add it again in case it still is duplicated.*

Example:

```
PositionCode [position=3] = HMAC(IDn||3, KRCG), truncate to the last 4 digits.
max_positions=30.
While (PositionCode[3] already exists){
       position=position+max_positions=33.
       PositionCode [position=3] = HMAC(IDn||33, KRCG), truncate to the last 4 digits.
}
```

12. Create list F:{B_{id}←→{$PartyCode_i$, $PositionCode_i$, EmptyVoteCode}}, or F:{B_{id}←→{$AnswerCode_i$, EmptyVoteCode}} in case of referendums ← The codes $PartyCode_i$, $PositionCode_i$ should be still associated to the original Party and Position names (in case of referendums, the same for the codes $AnswerCode_i$ and the answer names).
13. Encrypt the list with the public key of the printer service: $E_{Pprinter}(F)=F'$.
14. Send list F' to the printing service.
15. The values {$PositionCode_i$, $PartyCode_i$, $AnswerCode_i$, $PartyReturnCode_i$, and $CandidateCode_i$, $AnswerReturnCode_i$} mapped to the real names are used in the next calculations (see section 1.2.1.3).

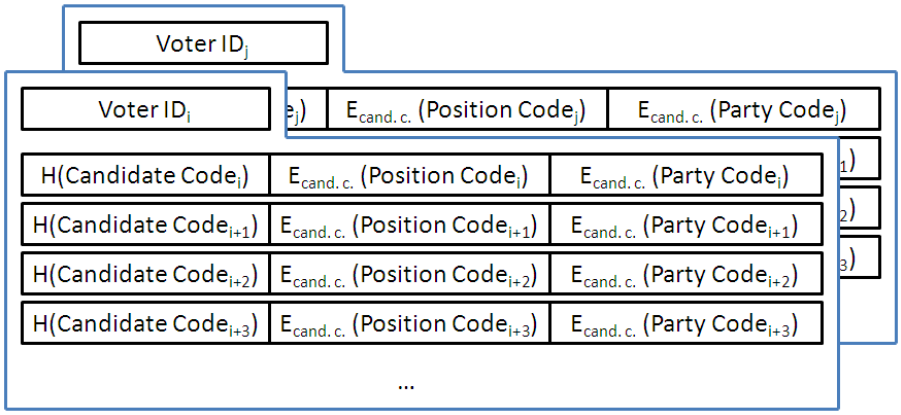
1.4.1.2 Candidate, Party and Answer Return Codes: translation to Position Codes, Party Codes and Answer Codes

To represent the Candidate Return Codes (or Candidate Codes) as positions of the party lists during the voting phase, the Party Return Codes as Party Codes, and the Answer Return Codes as Answer Codes, we use a table stored in the RCG (in the election configuration phase) to map the first type of codes to the second type.

In order to map the Candidate Codes to the Position Codes, the Party Return Codes to the Party Codes, and the Answer Return Codes to the Answer Codes, a hash of each possible Candidate Code, Party Return Code, or Answer Return Code will be stored in the table, connected to the code corresponding to the position of the candidate in the party list, to the PartyCode, or to the AnswerCode. The hashes of the Candidate Codes will also be connected to the PartyCodes, so that, when a voter chooses candidates from alternative lists, the PartyCode can be obtained to be sent by SMS jointly with the position of the candidate. For security issues, the position and party codes connected to the hash of the candidate code will be encrypted using the candidate code as a symmetric key. The same will be done with the party return codes and the party codes, and with the answer return codes and answer codes. Therefore, since the RCG cannot calculate the Candidate Code, the Party Return Code, or the Answer Return Code on its own (it is a process shared between the voter, the VCS and the RCG), it cannot access to that information until it has to be sent to the voter.

An example of the table stored in the RCG is shown in the next figure:

An example of the table stored in the RCG is shown in the next figure:



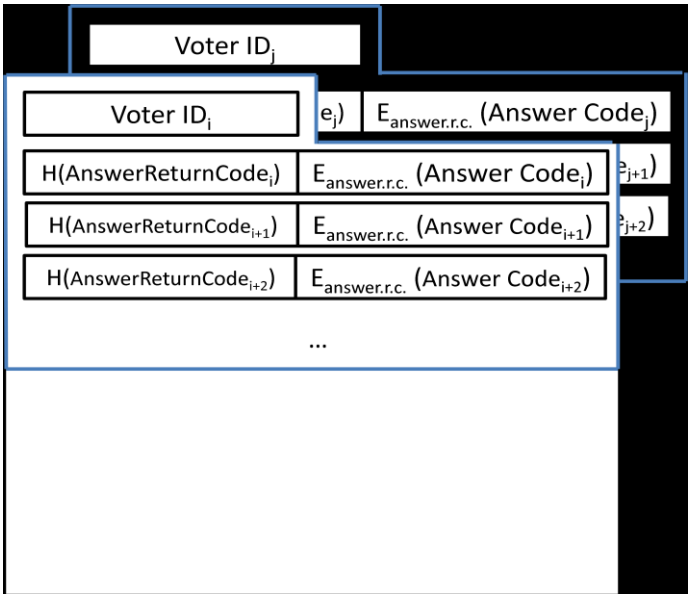
Note: *the voter identification ID_n is not stored in the table (since the RCG itself has not these identities bind to the Return Codes). The VoterID value is just included in this image to illustrate that all the hashes of all the candidate codes calculated for all the voters are inputs in the table.*

The hash function will be a SHA-256 algorithm, where its output will be trunkated to 128 bits, and the position and party codes are encrypted using the AES-128 symmetric algorithm. Each entry in the table has a size of 128x3 bits= 48 bytes.

Thus, when the RCG calculates a Candidate Code, it searches it in the table to retrieve the Position Code and also decrypts the Party Code in order to verify that is a valid candidate from a specific party. When the RCG calculates a Party Return Code, it searches it in the table to retrieve the related Party Code (which must be decrypted using the hash of the calculated Party Return Code as a key).

Then, the Position Codes and Party Codes (decrypted) are sent to the SMS gateway instead of the Candidate Codes and Party Return Codes (originally calculated).

In case we have a referendum, the table could be like this:



And the process will be the same explained above for the Party Return Codes.

Computation steps in RCG' (Election Configuration) Part 2: Creation of RCG's databases.

16. For all the CandidateCodes and Party Return Codes Codes associated to each B_{id}, compute:

    H(CandidateCodei)=last_128_bits_of(Hash(CandidateCodei)) ← The value of the hashes should be still associated to the original candidate name.

    H(PartyReturnCodei)=last_128_bits_of(Hash(PartyReturnCodei)) ← The value of the hashes should be still associated to the original party name.

    H(AnswerReturnCodei)=last_128_bits_of(Hash(AnswerReturnCodei)) ← The value of the hashes should be still associated to the original party name.

17. For each CandidateCode associated to each B_{id}, compute:

    a. $P=Encryption_{CandidateCode}(PositionCode_i)$, $Q=Encryption_{CandidateCode}(PartyCode_i)$
    using AES-128 encryption and the CandidateCode as a symmetric key (taking the last 128 bits), where the Party Code is the party the specific candidate is affiliated to and the Position Code is the position of the specific candidate in the party list.
    The same Position Code will be connected to different candidates in different parties, and the same Party Code will be connected to the candidates of the same party.

18. For each PartyReturnCode associated to each B_{id}, compute:

    b. $Py=Encryption_{PartyReturnCode}(PartyCode_i)$, **using AES-128 encryption and the PartyReturnCode as a symmetric key (taking the last 128 bits)**, where the Party Code is on top of the list of Candidate/Position Codes belonging to that party.

19. For each AnswerReturnCode associated to each B_{id}, compute:

c. $Py=Encryption_{AnswerReturnCode}(AnswerCode_i)$, **using AES-128 encryption and the AnswerReturnCode as a symmetric key (taking the last 128 bits)**.

20. Store in a table the values:

$H(PartyReturnCode_i) \leftrightarrow Encryption_{PartyReturnCode\_i}(PartyCode_i)$
$H(CandidateCode_i) \leftrightarrow Encryption_{CandidateCode\_i}(PositionCode_i), Q=Encryption_{CandidateCode\_i}(PartyCode_i)$.
$H(AnswerReturnCode_i) \leftrightarrow Encryption_{AnswerReturnCode\_i}(AnswerCode_i)$

Once the relationship between CandidateCode, PositionCode, PartyReturnCode and PartyCode has been stablished, their relationship with real names is broken (real names will be erased or won't be saved). The same holds for AnswerCodes.

21. Store the table in the RCG.

1.4.1.3 The Special Case of the EmptyVoteCode

Since the encrypted votes have a fixed length, some encrypted choices might represent an empty selection if the voter has not selected the maximum number of options. This empty selection is represented as a Ci=1 in the vote. In this case, the value of the preliminary calculation of the Return Code in the RCG during the voting process is

$$Ci' = 1^{si} \bmod p = 1.$$

Since the Return Codes are intended to represent the voter voting options, in case this value is found, no return code is calculated. Only Return Codes related to non-void voting options are sent to the voter.

However, in case an empty vote is submitted, no Return Codes would be calculated, and we need something to put in the SMS sent to the voter. In this case, only a Return Code is generated and submitted to the SMS gateway, the EmptyVoteCode:

$$EmptyVoteCode=HMAC(1||IDn, KRCG)$$, **and truncate to the last 4 digits**.

1.4.1.4 Printing the Voting Cards

After the Return Code generation in the VCS and the RCG, the printing service receives two lists:

The list $\mathbf{F}:\{B_{id} \leftrightarrow \{PartyCode_i, PositionCode_i, EmptyVoteCode\}\}$, or $\mathbf{F}:\{B_{id} \leftrightarrow \{AnswerCode_i, EmptyVoteCode\}\}$, optionally encrypted with the printing service public key, contains the Return Codes representing parties and positions in the party lists bind to the Voting Card identifier Bid. These Return Codes are associated to the original party and position names, so that they can be easily printed in the Voting Cards.
During the printing procedure, the Party and Position names, or answer texts, are printed jointly with their

respective Return Codes in the Voting Card. Once printed, each card is put into an envelope and the correspondent Bid is printed outside of it.

The list **B**:{ID_n ←→ B_{id}}, optionally encrypted with the printing service public key, is used to connect the Voting Card identifiers with real voters. The envelope containing each Voting Card is put into another envelope and the correspondent IDn connected to the Bid of the inner envelope is printed outside.

1.4.2 Representation of the Voting Options

1.4.2.1 Pre-election

Mapping the voting options

The voting options are represented by prime numbers.

In order to provide a secure encryption of the voting options, the prime numbers which represent them must be selected using the following procedure:

- Test the Legendre symbol of the prime numbers **in increasing order** and make two groups (one containing numbers with quadratic residue and another containing numbers with non-quadratic residue):
    - **After** the ElGamal public parameters {p, q, g} have been selected, test for each prime number if $v^q \bmod p = 1$ or not (where v is the prime number).
    - If the test is true, put the prime number in the first group. If not, put the prime number in the second group.
- Choose the group where there is the number 1.
- The primes from that group will represent the voting options.

For each group of generated ElGamal public parameters, two lists of primes (following this procedure) are created before the election: one list is stored (the one with number 1), and the other is discarded. Therefore, in the pre-election configuration phase, the group of primes that represent the voting options is defined after choosing the public parameters.

Testing of the voting options

Since the voting options chosen by a voter and encrypted individually are multiplied before the mixing process in order to get only one encryption per voter, it is important to evaluate if the product of the prime numbers 'inside' the individual encryptions is bigger than *q* (ElGamal parameter) or not. In case this product is bigger than *q*, the vote will not be successfully decrypted after the mixing. Therefore, when choosing the prime numbers at the Election Configuration stage, it is important to test if the product of the *N* prime numbers (where *N* is the maximum number of voting options that a voter can choose in a specific election) is bigger than *q*.

Depending on the type of election, this test will be slightly different:

Local and Municipal Elections

Being *nP* the number of parties, and *nCP* the number of candidates per party, *nC=nP x nCP* is the total number of candidates in the election. Therefore, we will need a list of primes to represent these *nC* candidates plus the *nP* parties ranging from *1* to the prime *nC+nP+1*.

The worst case when checking if the product of voting options is bigger than *q* is when the voter selects the *N* (max nº of voting options a voter can select) bigger prime numbers. That is, prime numbers from (*nP+nC+1)-N* to *nP+nC+1*.

TEST IF: the product of prime numbers from (*nP+nC+1)-N* to *nP+nC+1*, for a specific set of "possible voting options" (group of prime numbers) and ElGamal parameters is bigger than *q*.

- If not, output OK.
- If it is bigger, output the value = product {from (*nP+nC+1)-N* to *nP+nC+1*} /q, to know in how many blocs this set of options should be divided in order get a product that fits in *q*.

Parliamentary elections (preferential)

[party + (candidates, preferences)]

Being *nP* the number of parties, and *nCP* the number of candidates per party, we do not need to work with the value *nC=nP x nCP* = total number of candidates in the election, since in this case we represent the candidates with a code/prime number that informs about their position in a party list, and this code is the same independently of the party (see the following section!). Therefore, we will need a list of primes to represent these *nCP* positions (or candidates per party) plus the *nP* parties, ranging from *1* to the prime *nCP+nP+1*. Specifically, from this list, we will use the first *nCP+1* primes to represent the positions (or candidates) and the "no selection" or number one, and the last (and bigger) *nP* primes to represent the candidates.

Also, we will use an extra value for the preferences, which will be an integer in the range of {0,1, …, (nCP+1)} (e.g. if a party list has 20 candidates, the number of prime numbers to represent the candidates will be 20, and the number of integers to represent the preferences is 22: 0 for no preference, 1 to delete a candidate, and numbers {2..21} to assign preferences from 1 to 20).

The worst case when checking if the product of voting options is bigger than *q* is when:

- The voter selects the party represented with the higher prime number (the *1+nCP+nP*thprime in the list).
- The voter assigns to the positions represented by bigger prime numbers the higher preferences.

TEST IF: the product of the $1+nCP+nP^{th}$prime number in the list by the prime numbers from 2nd position to $nCP+1^{th}$ position (in the prime number list), raised each one to a value from 2 to nCP +1, for a specific set of "possible voting options" (group of prime numbers) and ElGamal parameters, is bigger than q.

e.g. if a party list has 20 candidates, we multiply the prime number representing the party list (we suppose is the higher prime number) by the second prime of the list raised to number one, the third prime of the list raised to number two, …, the 21^{th} prime of the list raised to number 21. Then, we compare if the resulting value is bigger than q or not.

- If not, output OK.
- If it is bigger, output the value number of blocs this set of options should be divided in to get a product that fits in q for each block.

The idea is to perform this test at the Election Configuration stage, so that the groups of ElGamal parameters {p, q, g} ←→ Assigned prime numbers for voting options (after the quadratic residue test) which do not accomplish the requirements are deleted from the list of possible combinations.

1.4.2.2 Voting phase

Order of the encrypted voting options

Each selected voting option is initially encrypted individually, instead of encrypting all the voting options together. This is required to generate the individual Return Codes for each voting option. Therefore, the vote is initially composed by a set of encrypted voting options.

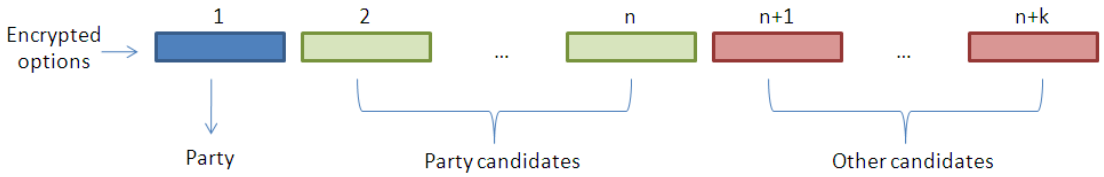
The selected vote options are ordered as follows:

- The first encrypted choice is the code of the selected party.

- Then, the next choices (up to the maximum number of candidates of the larger party list) are the ones belonging to the selections of candidates of the selected party (candidates for which the voter gives more weight). These selections are ordered according to their position in the list.

- Finally, the last encrypted choices belong to additional candidates from other parties than the one the voter has voted for (up to the number of candidates the voter can select). These selections have no order.

The next figure represents how the set of selected vote options is ordered:



Parliamentary elections

In Parliament Elections preferential vote is supported. When a voter selects a party, she can optionally assign different priorities to the candidates of this party or erase a candidate from the party list.

In order to support preferential vote the cryptographic protocol considers the following components:

1. Party code: each party is represented by a unique prime number

2. Position code: the position of a candidate in a party list is represented by a unique prime number. This prime number is related to a position to the list and all will be different from the party code assigned primes. This prime is independent of the party and therefore, the same position in different lists will have the same.

3. Preference: there are as many preference values as candidates in a party plus one. The extra one is related to the option of erasing a candidate from a party list. These preferences are exponents to which each position is raised according to the preference assigned by the voter to the candidate of the selected position. E.g., for the position 1 and assuming a maximum number of 20 selections allowed:

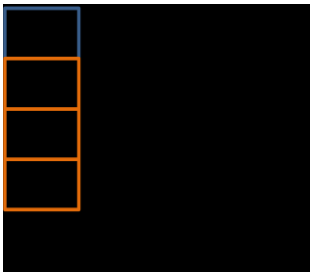
   p_1^3 means that voter assigned to position 1 a priority of 2.
   p_2^{21} means that voter assigned to position 2 a priority of 20.
   p_3^1 means that voter deleted candidate of position 3.
   $p_3^0 = 1$ means that voter did not assign any preference to the candidate of this position

The position codes *C* are the first *m* prime numbers (suitable to be encrypted using the ElGamal cryptosystem configured for semantic security), where *m* is the maximum number of possible candidates selectable in a party list.

The party codes *P* are the prime numbers from the $(m+1)_{th}$ to the $(m+k)_{th}$ (suitable to be encrypted using the ElGamal cryptosystem configured for semantic security), where *k* is the number of possible parties.

The preferences *PR* are integers that represent the preference level (e.g. *PR=3* for preference level 2).

Then, a party list is represented as:



The encryption of the selected options, [party + (candidates, preferences)] is done by the voting application in the following way:

- ElGamal encryption of the party code:

  $c_p=(P.h_{EB}^{rp}, g^{rp})$, where r_p is a random number in Z_q.

- ElGamal encryption of the candidate positions and the assigned preferences:

  $c_i=((C_i)^{PRj}.h_{EB}^{ri}, g^{ri})$, for i=$1...m$ where m is the number of possible candidates and j is the number of possible preference codes.

  The exponents r_i are random numbers in Z_q.

  If a candidate has not an assigned preference PR_j is set to 0 ($PR_j=0$), so the encryption would be $c_i=(1.h_{EB}^{ri}, g^{ri})$.

1.4.3 Key Management

1.4.3.1 Key Definition

Modules

The following modules compose the voting platform or interact with them. Therefore, they will need any type of key:

- Functional components (eVoting):
  - Front End.
  - Authentication Service.
  - Cleansing.
  - Mixing.
  - Counting.
  - VCS.
  - RCG.
  - eVoting Administration module.
  - Issuing Points in controlled electronic voting environments.
  - Settlement.

  - Reporting.
  - Audit.
- Functional components (pVoting):
  - Administrative System.
  - RBAC.
  - Electoral Roll.
  - pCounting centres [transfer].
  - pCounting centres [scanning].
  - Election sites.
  - Machines in municipal networks.
- People:
  - Voters.
  - Administration Board.
  - Electoral Board.
  - Electoral Officers: in charge of the pCounting centres and the Issuing Points in controlled electronic voting environments, etc.
  - Electoral Officers in charge of the Settlement module.

Some functional components considered in the key management definition may be allocated in the same machine. SSL and log keys would be then shared, but not the application keys.

Key Types

The modules operating in the voting platform need different keys from different purposes. In this document, we will distinguish between:

- Signing keys:
  - **Server keys** (for authentication in SSL/TLS connections): these keys are specific for each machine.
  - **Application keys** (for signing application data): these keys are specific for each component.
  - **Log keys** (for signing immutable logs): these keys are specific for each machine, although they are used to sign data at an application level. Logs from different applications running in the same machine will be aggregated in a single immutable log chain and signed with the same log key pair.
  - **User keys** (for signing application data): these keys are handled by physical persons.

- Encryption keys:
  - **Application keys** (for encrypting/decrypting application data): these keys are specific for each component.
  - **User keys** (for encrypting/decrypting application data): these keys are handled by physical persons.

These different-purpose keys are also valid for different events:

- The same Server keys and Log keys will be generally **used through different election events**.
- Application and User keys will be **generated for each specific election event**.

In order to distinguish between these two uses (one election or several elections), these keys may be certified by different local CA's.

Keys needed at each module:

Modules	Generation	Common name (certificate)	Certificate filename (public key)	P12 filename (private key)	Algorithm	Key usage
eVote Front End						
SSL keys	HSM (cert by Trusted CA)	domain	Pa_ssleVoteFE	Sa_ssleVoteFE	RSA	SSL connections (server)
Authentication Front End						
SSL keys	HSM (cert by Trusted CA)	domain	Pa_sslAuthFE	Sa_sslAuthFE	RSA	SSL connections (server)
Admin Front End						
SSL keys	KMS	domain	Pa_sslAdminFE	Sa_sslAdminFE	RSA	SSL connections (server)
Authentication Service						
Application signing keys	HSM	Auth_appSign_eventID	Ps_appAuth_eventID	Ss_appAuth_eventID	RSA	Signs the Authentication Token needed get a voter identified and cast a vote.
	External entity (certified by trusted CA)	KRD_Sign	Ps_KRD	Ss_KRD		Signs communications with ID-Porten
Log signing keys	HSM	Auth_logSign	Ps_logAuth	Ss_logAuth	RSA	Log signing.
KMS						
Application signing keys	KMS	KMS_appSign_eventID	Ps_appKMS_eventID	Ss_appKMS_eventID	RSA	Signs the intermediate configuration data going to other modules, like VCS' and RCG'.

		KMS_codeSign	Ps_codeKMS	Ss_codeKMS	RSA	Signs some code which has to be signed (e.g. applets)
Log signing keys	KMS	KMS_logSign	Ps_logKMS	Ss_logKMS	RSA	Log signing.
VCS' (representant)						
Application signing keys	KMS	VCSrep_appSign_eventID	Ps_appVCSrep_eventID	Ss_appVCSrep_eventID	RSA	Signs its output data (in the configuration stage).
Log signing keys	KMS	VCSrep_logSign	Ps_logVCSrep	Ss_logVCSrep	RSA	Log signing.
RCG' (representant)						
Application signing keys	KMS	RCGrep_appSign_eventID	Ps_appRCGrep_eventID	Ss_appRCGrep_eventID	RSA	Signs its output data (in the configuration stage).
Log signing keys	KMS	RCGrep_logSign	Ps_logRCGrep	Ss_logRCGrep	RSA	Log signing.
Cleansing						
Application signing keys	KMS	Cleansing_appSign_eventID	Ps_appCl_eventID	Ss_appCl_eventID	RSA	Signs the cleansed ballot boxes which go to the mixing.
Log signing keys	KMS	Cleansing_logSign	Ps_logCl	Ss_logCl	RSA	Log signing.
Mix-nodes						
Application singing keys	KMS	Mix_node_[x]_appSign_eventID	Ps_appMix[x]_eventID	Ss_appMix[x]_eventID	RSA	Each mix-node signs its output data: mixed votes and audit proofs.
Log signing keys	KMS	Mix_node_[x]_logSign	Ps_logMix[x]	Ss_logMix[x]	RSA	Log signing.
Encryption application keys	KMS	NA	NA	Ke_Mix[x]_eventID	RSA	Encrypts mixing data used for audit.
Counting						
Application signing keys	HSM	Counting_appSign_eventID	Ps_appCount_eventID	Ss_appCount_eventID	RSA	Signs the results of

						eVote counts.
Log signing keys	HSM	Counting_logSign	Ps_logCount	Ss_logCount	RSA	Log signing.
VCS						
Application signing keys	HSM	VCS_appSign_eventID	Ps_appVCS_eventID	Ss_appVCS_eventID	RSA	APP signing key: Signs its output data: ballot box, crypto proofs for the RCG.
Log signing keys	HSM	VCS_logSign	Ps_logVCS	Ss_logVCS	RSA	Log signing.
Encryption application keys	KMS (VCS') (Secret Sharing)	NA	Pe_elgVCS_eventID	Se_elgVCS_eventID *(Se_elgVCS_sh[x]_eventID for shares)*	ElGamal	To generate the partial values of Return Codes.
	KMS (VCS')		NA	Ke_VCS_eventID	RSA	To generate the partial values of Return Codes.
	HSM		Pe_rsaVCS_conf_eventID	Se_rsaVCS_conf_eventID		To bring encrypted configuration data from the KMS.
RCG						
SSL keys	HSM	RCG_SSL	Pa_sslRCG	Sa_sslRCG	RSA	SSL connections (server)
Application signing keys	HSM	RCG_appSign_eventID	Ps_appRCG_eventID	Ss_appRCG_eventID	RSA	APP signing key: Signs its output data: voting receipt list, voting receipts.
	External entity (certified by trusted CA)	KRD_Sign	Ps_KRD	Ss_KRD		Signs communications with ID-Porten
Log signing keys	HSM	RCG_logSign	Ps_logRCG	Ss_logRCG	RSA	Log signing.
Encryption application keys	KMS (RCG') (Secret Sharing)	NA	Pe_elgRCG_eventID	Se_elgRCG_eventID *(Se_elgRCG_sh[x]*	ElGamal	To generate the final

				_eventID for shares)		values of Return Codes.
	HSM		NA	Ke_RCG_eventID	Sym	
			Pe_rsaRCG_conf_eventID	Se_rsaRCG_conf_eventID	RSA	To bring encrypted configuration data from the KMS.
	May be external		NA	Ke_idporten	Sym	To exchange data with the phone number service.
Issuing Points						
SSL keys (may be the same for the Pollbook application)	KMS	Issuing_Point_Mun[x]_SSL	Pa_sslIPMun[x]	Sa_sslIPMun[x]	RSA	SSL connections (client)
Application signing keys	KMS (until Buypass is integrated)	Issuing_Point_Mun[x]_appSign_eventID	Ps_appIPMun[x]_eventID	Ss_appIPMun[x]_eventID	RSA	Sign the voter smartcard to authorize a voter to get authenticated in the Auth. Service from a controlled environment.
Log signing keys	KMS	Issuing_Point_Mun[x]_logSign	Ps_logIPMun[x]	Ss_logIPMun[x]	RSA	Log signing.
Log immutabilizator						
Log signing keys	KMS	Immutabilizator[x]_logSign	Pa_logImm[x]	Pa_logImm[x]	RSA	Log signing.
Admin						
Application signing keys	KMS	Admin_appSign_eventID	Ps_appAdmin_eventID	Ss_appAdmin_eventID	RSA	Signs its output data, e.g. EML configuration file.
Log signing keys (for all the components in Admin)	KMS	Admin_logSign	Ps_logAdmin	Ss_logAdmin	RSA	Log signing.
BigIP						
SSL keys	BigIP (certified by Trusted CA)	domain	Pa_sslBigIP	Sa_sslBigIP	RSA	SSL connections (server)
pVoteAdmin						

SSL keys	KMS	pVoteAdmin_Mun[x]_SSL	Pa_sslpVoteAdmMun[x]	Sa_sslpVoteAdmMun[x]	RSA	SSL connections (client).
Log signing keys (for all the components of eCounting of pVotes)	KMS	pVote_Mun[x]_logSign	Ps_logpVoteMun[x]	Ss_logpVoteMun[x]	RSA	Log signing.
Voting terminals in Polling stations						
SSL keys	KMS	Polling_Stat_Mun[x]_SSL	Pa_sslPSMun[x]	Sa_sslPSMun[x]	RSA	SSL connections (client)
Admin Client PCs						
SSL keys	KMS	Municipal_[x]_SSL	Pa_sslMun[x]	Sa_sslMun[x]	RSA	SSL connections (client)
Several servers						
TPM keys	TPM	[module_name]_[server_id]_TPM	Ps_tpm[module][server_id]	Ss_tpm[module][server_id]	RSA	Used to sign audit results from the specific server.
Voters						
User signing keys	KMS	Voter_Hash(SSN)_Sign_eventID	Ps_voter[x]_eventID	Ss_voter[x]_eventID	RSA	Used to sign the votes.
Administration Board						
User signing keys	KMS (Secret Sharing)	AB_Sign_eventID	Ps_AB_eventID	Ss_AB_eventID *(Ss_AB_sh[x]_eventID for shares)*	RSA	Used to sign election configuration and decrypted votes.
Electoral Board						
User encryption keys	KMS (Secret Sharing)	NA	Pe_EB_eventID	Se_EB_eventID *(Se_EB_sh[x]_eventID for shares if there are any)*	ElGamal	Used to decrypt the votes.
Electoral Officers						
User signing keys	Buypass	EO_[x]_Sign	Ps_EO[x]	Ss_EO[x]	RSA	Sign several things: voter smartcard to authorize a voter to get authenticated in the Auth. Service

						from a controlled environment (in the Issuing Points); pVotes results to be sent to the settlement module.
Electoral Officers in Settlement						
User signing keys	KMS (Secret Sharing)	Settlement_user Sign_eventID	Ps_Set_eventID	Ss_Set_eventID *(Ss_Set_sh[x]_eventID for shares)*	RSA	Sign the election results.

1.4.3.2 Phases:

The following phases will be described:

- Key generation.
- Key distribution.
- Key use.
- Key validation.
- Key cancelation.

Key generation

Depending on the type of keys and their users, the cryptographic keys will be generated by different means:

- **Key Management Service** (managed by Scytl): the KMS is a service composed by a **local CA** working offline and a Key Generator. **This local CA may be certified by an external CA (GlobalSign).** Using as input the number of needed key pairs, the type, and the Common Names to be specified in the digital certificates, this service generates PKCS#12 containers containing the key pairs and the digital certificates, and random passwords which will open them.

  Passwords may be also preset before this generation, like in the case of PKCS#12's containing the signing keys for voters, which will be sealed with their SSNs.

  Especially sensitive keys may be directly generated, certified, split in shares and stored in smart cards, like the Administration Board signing keys and the Electoral Board encryption keys.

  The following keys are generated using this system:

  - All the server keys, less for the Front End and for the modules with HSMs (specified below).
  - All the application signing keys, less for the modules with HSMs (specified below).
  - All the log signing keys, less for the modules with HSMs (specified below).

- The user signing keys for the voters, for the Administration Board, and the user signing keys for the Electoral Officer in case an alternative service like Buypass is not available.
- The user signing keys for Electoral Officers in charge of the Settlement module: special requirements.
- All the encryption application keys.
- All the encryption user keys.

- **KMS + Trusted CA** (e.g. Verisign): some keys may be generated in the KMS, but need to be certified by a **well-known CA** (one that is already configured in the voters' browser). In this case, the KMS will generate the CSR (Certificate Signing Request), which will be sent to the trusted CA to get a signed digital certificate. After that, the KMS will compose the PKCS#12 containers as in the previous case.

  The following keys are generated using this system:

  - Server keys for the Front End machines.

- **HSMs** (Hardware Security Module)**:** HSMs are used in several modules of the election architecture, in order to manage/generate sensitive keys used to decrypt votes, digitally sign data, and manage authenticated communications. In these modules, the HSM will be in charge of generating the needed keys instead of the KMS. Then, CSRs will be generated to obtain digital certificates certified by the **local CA**. The HSMs output CSRs, which are transported to the CA in order to create the digital certificates. The digital certificates are then returned to the HSMs.

  The following keys are generated using this system:
  - The server keys for the modules with HSMs.
  - The application signing keys for the modules with HSMs.
  - The log signing keys for the modules with HSMs:
    - Authentication Service.
    - Counting.
    - VCS.
    - RCG.

- **Buypass authentication/signature functionalities:** when possible, this service will be used for managing user signing keys. Buypass signing keys are generated in the smart cards that will be issued to the users, and certified by their specific service.

  The following keys are generated using this system when available:

  - The user signing keys for the Electoral Officers.

Key generation / Keys	KMS	KMS + Trusted CA	HSMs	Buypass
Server Keys	All modules less Front Ends, AS, VCS, RCG and Counting.	Front Ends	AS, VCS, RCG and Counting modules.	
Application signing keys	All the modules less for AS, VCS, RCG and Counting.		AS, VCS, RCG and Counting modules.	
Log signing keys	All the modules less for AS, VCS, RCG and Counting.		AS, VCS, RCG and Counting modules.	
User signing keys	Voters, Administration Board, EOs in charge of Settlement.			Electoral Officers (less those in charge of the Settlement).
Encryption application keys	All the modules: VCS, RCG			
Encryption user keys	Electoral Board.			

Key distribution

Five means of key distribution will be used:

- **Server key distribution:** all the server keys (stored in PKCS#12s), less those already generated and stored in HSMs, will be distributed manually to the modules (e.g. distributed via removable media). In case of the following modules, the same server keys are used for all the machines in the same municipality, and are managed by a system administrator designed for the specific municipality:
  - Issuing Points in controlled electronic voting environments.
  - pCounting centres [transfer].
  - Election sites.
  - Machines in municipal networks.

- **Database Keystores** (all less server keys)**:** these keystores will be used in modules situated in data centres. They are databases containing the PKCS#12's with the private keys of the modules where they are installed, and all the **valid** digital certificates of all the modules (we will use them as **white lists**).

  In case a module has a key pair stored in a HSM, its database keystore input will point to the local direction of the HSM.

  Database keystores will be installed in the following modules:

  - Authentication Service: containing AS's PKCS#12's, voters' PKCS#12's, and digital certificates needed for the verification of signatures.

- Cleansing: containing Cleansing's PKCS#12's and digital certificates needed for the verification of signatures.
- Counting: containing Counting's PKCS#12's and digital certificates needed for the verification of signatures.
- VCS: containing VCS's PKCS#12's and digital certificates needed for the verification of signatures.
- RCG: containing RCG's PKCS#12's and digital certificates needed for the verification of signatures.
- eVoting Administration module: containing eVoting Administration module's PKCS#12's and digital certificates needed for the verification of signatures.
- Settlement: containing Settlement's PKCS#12's and digital certificates needed for the verification of signatures.
- Administrative System: containing Administrative System's PKCS#12's and digital certificates needed for the verification of signatures.
- Electoral Roll: containing Electoral Roll's PKCS#12's and digital certificates needed for the verification of signatures.
- RBAC: containing RBAC's PKCS#12's and digital certificates needed for the verification of signatures.

- **Online Keystore** (all less server keys)**:** this keystore will be used by modules situated in several places, where the manual deployment is too complex. This is an online database (with access control) containing the PKCS#12's with the private keys of this modules, and all the **valid** digital certificates of all the modules (**white lists**).

  The online keystore will be accessed and used by the following modules:
  - Issuing Points in controlled electronic voting environments.
  - pCounting centres [scanning]. In case the scanning modules are offline, they will access to the online keystore through an online machine of the pCounting center (e.g, transfer module).
  - Election sites.
  - Machines in municipal networks.

- **Key roaming:** a key roaming mechanism is implemented in order to deliver the signing keys to the voters. The PKCS#12's with these signing keys will be stored in the Authentication Service database and sent to the voters (jointly with digital certificates for signature verification) from there, at the time of authentication.

- **Smart card delivery:** smart cards with cryptographic keys are delivered to some specific users:
  - Administration Board.
  - Electoral Board.
  - Electoral Officers in charge of the Settlement module.

- Electoral Officers in charge of the pCounting modules and the Issuing Points in controlled electronic voting environments, etc. In this case, the smart cards will be provided by Buypass.

Distribution / Keys	Manual	Database keystores	Online keystores	Key roaming	Smart cards
Server keys (less for modules with HSMs)	x				
Other keys for machines in databases		x			
Other keys for machines in poll sites/other networks			x		
User keys for voters				x	
User keys for AB, EB, EO.					x

Key use

Once all the modules have their key pairs available, they also have to get the PIN codes to open the PKCS#12 containers. **The distribution of the PIN codes will be done manually for all the modules, less for the voters (the passwords are their SSNs), and the users with smartcards, where they can usually select their PIN.**

Therefore, the PIN codes for accessing server keys (less for modules with HSMs) and keys in database/online keystores are distributed manually. In case of the following modules, the PIN code distribution is managed by a system administrator designed for each specific municipality:

- Issuing Points in controlled electronic voting environments.
- pCounting centres [scanning].
- Election sites.
- Machines in municipal networks.

Database/online keystores work at an application level. Therefore, the PIN codes for the server keys must be introduced in the machines before accessing the keystores. The steps are the following:

1. Start up the machine.
2. The PIN code for the server key will be requested.
3. Insert the PIN code.
4. The machine starts running and gets blocked at some point.
5. The blocked machine asks for the PIN codes of the PKCS#12's in the keystore.
6. Insert the PIN codes.

7. The machine accesses the module database keystore or the online keystore and gets its PKCS#12's.
8. The machine opens the PKCS#12's with the PIN codes and keeps the keys in memory.

Key validation and certificate revocation

The database/online keystores will be used by the different modules to validate a digital certificate when validating a digital signature. When receiving a signed message from another module, the module asks to its keystore for the corresponding digital certificate to validate the signature.

In case a digital certificate is revoked due to private key compromise, a new key pair must be assigned to the specific module, and the white lists in the keystores must be updated:

- **Assigning a new key pair:**
  - In case of server keys, install it manually and reboot the server, so a new PIN code is requested when starting the machine.
  - In case of other keys, update the PKCS#12 input in the database or online keystore and insert the new PIN code in the module. In case of keys contained in smart cards, new smart cards with the new keys must be personally issued.

- **Updating the white lists:**
  - When a certificate is revoked all the keystores must be updated, marking the old certificate as revoked (specifying the time of revocation) and adding the new one.

1.4.3.3 TPM (Trusted Platform Module)

TPMs will be used in some modules as a Remote Attestation tool, in order to ensure that the initialization of the machines and applications is the one expected, and that no external agent is interfering with the correct performance of the applications.

To do this, at first a fingerprint of the processes that will be executed when starting the machine and executing the applications is generated. After that, when the voting process starts, the TPM generates a fingerprint of the processes that are being executed at each step, and sends it to a remote application where it will be compared with the first fingerprint, in order to check if they are what they are expected to be. The TPM uses a trusted key pair generated inside of the hardware module to sign the fingerprint.

TPM signing keys:

The way the TPM signing keys are handled can be the same than in the case of HSMs, since they are also generated inside a hardware module (section 3.1): The TPM generates a key pair inside of the module and outputs the public key signed by the private key. With this data, a CSR is generated and transported to the local CA, where a digital certificate is generated and returned to the TPM.

Also, the TPM can provide their own certificates in case that the endorsement keys (the key pair generated at the time of manufacturing the chip) are used to sign the fingerprints. In that case, a certificate issued by the manufacturer can be used, instead of using the local CA to generate it.

Attestation distribution:

The attestation fingerprints will be distributed to two different places: they will be included in the secure logs, and they will be sent to the central audit module for monitoring purposes.

Modules:

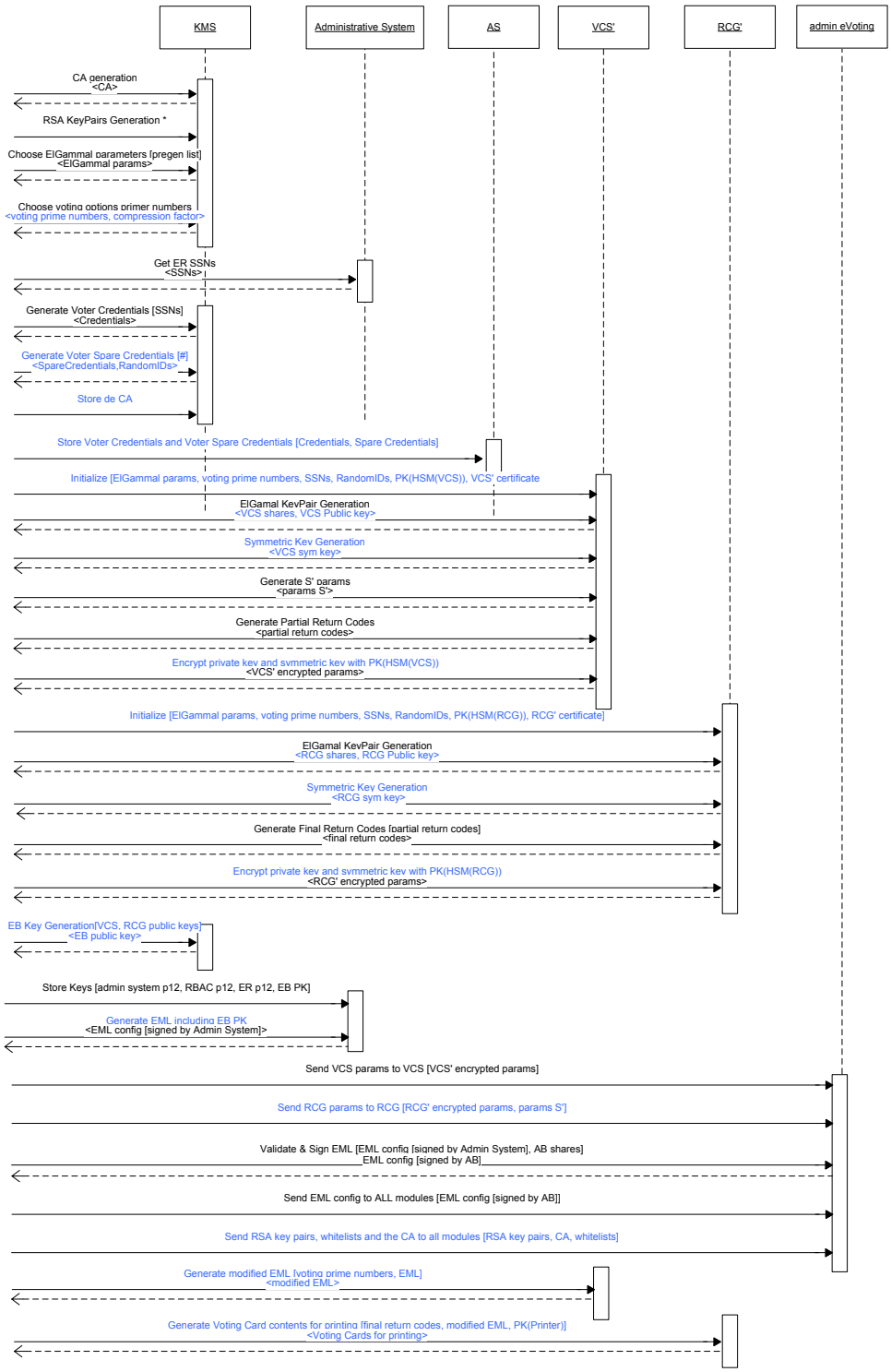
The modules which will have TPMs on the machines where they are executed are:

- Front End.
- Authentication Service.
- Cleansing.
- Mixing.
- Counting.
- VCS.
- RCG.
- Audit.
- Settlement.
- Reporting.
- Administrative System.
- RBAC.
- Electoral Roll.
- pCounting centres [transfer].

1.5 Process overview

1.5.1 Election configuration process

The following sequence diagram depicts the interaction between the different modules during the election configuration phase:



1.5.1.1 Specific Notation

Before explaining the steps of the election configuration phase, some specific notation must be explained:

- CA: Certification Authority.
- ER SSNs: Social Security Numbers obtained from the Electoral Roll.
- PK: Public Key.
- HSM: Hardware Secure Module.
- TPM: Trusted Platform Module.
- RBAC: Roll-Based Access Control.

1.5.1.2 Steps of the election configuration phase:

The steps presented in the sequence diagram are explained here with more detail.

Previous Steps

i. Pre-generation of a list of ElGamal public parameters: a list of several sets of ElGamal public parameters {p, q, g} is generated.

ii. Pre-generation of sets of voting options: for each group of ElGamal public parameters, two groups containing the prime numbers that are going to represent the voting options are defined.

Election Configuration Phase

1. **CA generation:** the Key Management Service generates an RSA key pair (for performing digital signatures) for the Certification Authority.
   Two different CAs may be generated, one for generating system certificates (for log signing keys and SSL) , and another for generating election certificates (for application signing keys and voter keys), since the certificates may have different validity periods.
   Also, an external CA may be used to certify the local CAs.
2. **RSA key pair generation:** the Key Management Service generates RSA key pairs (for performing digital signatures) for several modules defined in another document. These RSA key pairs are used for different purposes:
   - Server (SSL) keys
   - Application signing keys
   - Log signing keys

   The RSA key pairs are signed by the CA (using the CA private key). The private key of each key pair is stored in a PKCS#12 format and sealed with a PIN randomly generated. PINs must be configured in each module.

As a specific case, one RSA key pair is generated for the Administration Board. The private key is divided in shares, which are signed (by the private key itself) and stored in smartcards.

3. **RSA key pair certification for keys generated in HSMs and TPMs.**
   - **RSA key pair generation in HSM**: the Authentication Service, the VCS, the RCG and the Counting Service have HSMs assigned. RSA signing key pairs, and encryption key pairs (RSA or ElGamal) in case of VCS and RCG, are generated in the HSMs. The signing keys are sent to the Key Management System to be certified by the CA (Certificate Signature Request).
   - **RSA key pair generation in TPM**: all the servers have TPMs installed for remote attestation purposes. These TPMs generate RSA keys, which are certified or registered in the KMS.
4. **Set of ElGamal public parameters:** the ElGamal public parameters {p, q, g} are randomly chosen from the pre-generated list (in the Key Management Service).
5. **Set of voting options:** the Key Management Service chooses the set of prime numbers representing the voting options assigned to the selected ElGamal parameters. The file of voting options also contains the compression factor, which indicates in how many groups of voting options can be compressed a vote in the cleansing process.
6. **Obtaining the SSNs:** the Social Security Numbers of registered voters are obtained from the Electoral Roll module.
7. **Generating voter credentials:** RSA key pairs for the voters are generated in the KMS and certified by the CA. The private key of each key pair is stored in a PKCS#12 format and sealed with the SSN of the voter.
8. **Generating spare voter credentials:** RSA key pairs are generated in the for voters who might get registered during the election generated in the KMS and certified by the CA. The private key of each key pair is stored in a PKCS#12 format and sealed with a random PIN (since we do not know the SSNs).
9. **Store the CA/CAs:** after generating all the certificates, the CA/CAs are securely stored (i.e. in a smartcard).
10. **Storing voter credentials:** the voters' credentials and spare voter credentials are stored in the Authentication Service.
11. **Initialization of VCS':** once the ElGamal public parameters are defined, an isolated PC 'representing' the VCS datacenter can be initialized.
12. **Generation of VCS parameters:** the VCS' generates specific parameters on behalf of the VCS:
    - Symmetric key Kvcs.
    - Private voter parameters {si} and {g^si}=S'.
    - Return Codes': for all the voting options, and for all the voters (included spare voters), a partial calculation of the Return Codes is performed.
    - The VCS ElGamal key pair: the private key is divided in shares and stored in the EB members' smartcards, and the public key is stored in a file.
    - It is very important that the shares are labeled when stored in the smartcards with the name of the component they belong to (VCS in this case).

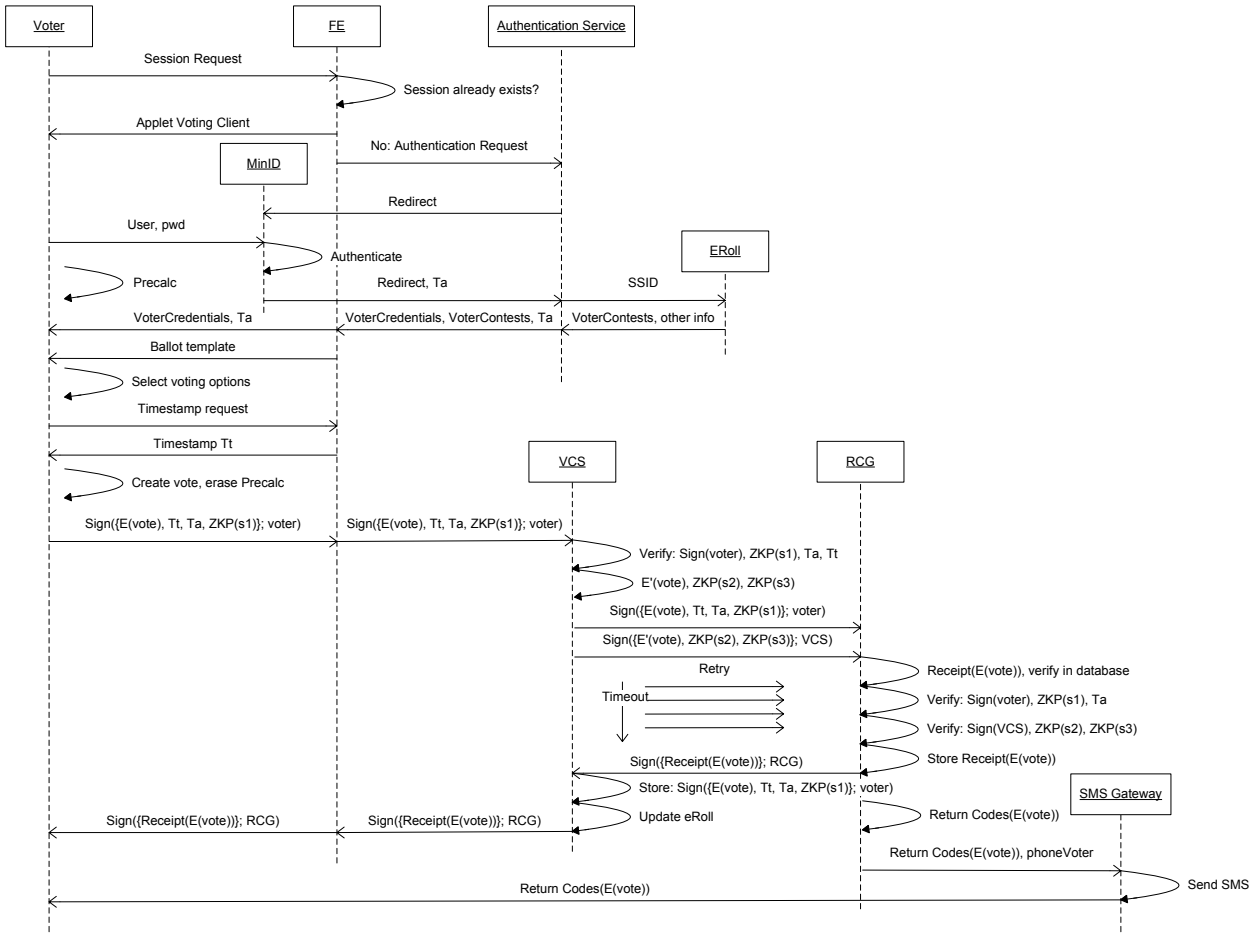
The ElGamal private key shares are encrypted jointly with the symmetric key Kvcs using the public key of the HSM in the VCS.

13. **Initialization of RCG':** an isolated PC 'representing' the RCG datacenter can be initialized to generate specific parameters for the RCG. The RCG' receives the partial calculation of the Return Codes (Return Codes') from the VCS'.
14. **Generation of RCG parameters:**
    - Symmetric key Krcg.
    - The RCG ElGamal key pair: the private key is divided in shares and stored in the EB members' smartcards, and the public key is stored in a file.
    - It is very important that the shares are labeled when stored in the smartcards with the name of the component they belong to (VCS in this case).
    - The final values of the Return Codes (also for spare voters).
    - The ElGamal private key shares are encrypted jointly with the symmetric key Krcg and the values {g^si} using the public key of the HSM in the RCG.
15. **Generation of EB ElGamal public key:** the public keys of the VCS' and the RCG' are used (in the KMS) to calculate the ElGamal public key of the Electoral Board.
16. **Obtaining the configuration EML:** the Key Management System sends the RSA key pairs (in a PKCS#12 format) of the Administrative System, RBAC and Electoral Roll to the Administrative System, which distributes them among the other modules. The KMS also sends the Electoral Board ElGamal public key to the Administrative System. This key is added to the Election Configuration EML document generated in the Administrative System, which is then digitally signed by this module and sent to the KMS.
17. **Validation and distribution of the configuration EML**: the Key Management System validates the RSA signature of the EML (by the Administrative System) and the configuration data. Once validated, the EML is digitally signed by the Administration Board and distributed to all the modules that need it.
18. **Generating the modified EML:** the VCS' receives the configuration EML and generates a modified EML where the names of parties and candidates (in the original EML) are mapped to the primes representing the voting options.
19. **Mapping the final RC to real names:** the modified EML is sent to the RCG', where the final Return Codes are linked to the real names of parties and candidates (until now, they were linked to the prime numbers representing the voting options.
20. **Generating the voting cards' contents:** the final return codes are formatted in the RCG' for being printed in the Voting Cards: they are shortened for easy readability, duplicates in the final Return Codes for each voter are checked and fixed (generating another final RC), and the file is translated into a printing format.
21. **Print Voting Cards:** the RCG' encrypts the groups {Return Codes, voter election identity} with the public key of the printer, and sends them to it (air-gapped connection), where they are printed in voting cards. **The relationship between the Return Codes and the voter election identity (SSN) must be secretly preserved until the printing and distribution of the Voting Cards.**

22. **Sending data to the VCS and RCG:** the contents generated and encrypted (or not) in the VCS' and RCG' are sent from the KMS to the VCS and RCG modules respectively. When received, the encrypted contents are decrypted by the HSMs and securely stored.
    (The S params are sent from the VCS' to the RCG)
23. **Send keys and certificates to the platform modules:** the PKCS#12's corresponding to the eVoting platform modules (Cleansing, Mixing, Counting…), the digital certificates, and the CA, are sent to the specific modules.

1.5.2 Voting process

The following sequence diagram depicts the interaction between the different modules during the voting process:



1.5.2.1 Specific notation

Before explaining the steps of the voting process, we will present here some specific notation used in the sequence diagram:

- Ta: Authentication Token, which contains the SSID of the voter (Social Security number).
- VoterCredentials: PKCS#12 which contains the private key of a voter (for performing digital signatures) encrypted with the hash of her SSID.
- VoterContests: contests in which the voter is authorized to vote in.
- E(vote): encryption of the voting options using the Electoral Board public key.
- Sign({m}; a): signature of the message m with the private key of the entity a.
- ZKP: Zero Knowledge Proof (Cryptographic primitive).
- E'(vote): the result of performing a partial decryption and a re-encryption over the encrypted voting options.

1.5.2.2 Steps of the voting process

The steps presented in the sequence diagram are explained here with more detail.

1. **Session request:** the voter accesses the eVoting website, sending a Session Request to the Front End.
2. **Voting Client:** after checking that the voter has Java and Javascript in her computer, the Voting Client Applet starts downloading in the voter side in background.
3. **Authentication request:** the Front End checks if there already exists an opened session for this voter. If not, it forwards the request to the Authentication Service**.**
4. **MinID identification:** the Authentication Service redirects the voter to the MinID application, where she inputs her user and password. At this time, Voting Client starts pre-calculating cryptographic values (for the encryption), until its buffer is full. The buffer size will be determined by the number of possible options in the ballot. MinID authenticates the voter and issues an Authentication Token (Ta), containing the SSID of the voter.
5. **Obtaining Voter Credentials:** the Ta is forwarded to the Authentication Service, which verifies it and reads the SSID in order to obtain the proper VoterCredentials and the elections the voter is authorized to vote in (from the Electoral Roll). The VoterCredentials and the Ta are sent to the Voting Client through the Front End, as well as the proper ballot template.
6. **Constructing and sending the encrypted vote:** the voter selects her chosen voting options, which are encrypted by the Voting Client. Also, a Zero Knowledge Proof based on the Schnorr Signature (ZKP(s1)) is attached to the encrypted vote. The voter accesses to her private key (to sign) from the VoterCredentials and digitally signs the encrypted options, the ZKP, the Tt (a timestamp requested to the F.E. for a timestamp Tt) and the Ta. In that moment, the precalculated values stored in the Applet buffer are erased. The signed vote is sent to the VCS through the F.E.
7. **Verifying the vote (VCS):** when the VCS receives the vote, it verifies:
   - The digital signature: checking that the author of the vote is an eligible voter.

- The ZKP(s1): in order to detect replications.
- Ta: in order to detect replications and verify that the voter has gone through the previous authentication process, the Ta is verified. If a Ta has been used once, a new vote containing this Ta will be rejected.
- Verify that the Tt matches the local time in the VCS (with a certain margin).

8. **Processing the vote (VCS):** if the verifications are successful, the VCS performs a partial decryption of the vote using its private key (for decryption). It also calculates two Zero Knowledge Proofs based on the Schnorr Signature in order to proof the correctness of this partial decryption (ZKP(s2), ZKP(s3)).
9. **Forwarding the vote and its partial decryption:** the VCS forwards the vote sent by the voter to the RCG, and also sends the partially decrypted vote and the ZKP's, digitally signed using its credentials. **Connection error tolerance:** If the VCS does not receive any response from the RCG (voting receipt), it tries to send the information two times more until a timeout signals the error of connection. Then, the voter is notified and requested to vote again. The unsent vote is stored in the log information.
10. **Verifying the information (RCG):** when the RCG receives that information, the first action is to calculate the voting receipt (hash of the encrypted vote) and compare it to the voting receipts stored in its database. If the voting receipt is already in the database, the RCG sends it to the VCS (in order to manage the possible "resends" of the VCS).

    If not, the RCG verifies from the vote:
    - The digital signature: checking that the author of the vote is an eligible voter.
    - The ZKP(s1): in order to detect replications.
    - Ta: verify that the authentication token has not been used before (RCG must maintain a table of received Ta's).

    The RCG also verifies the information calculated by the VCS:
    - The digital signature: checking that the author of the information is the VCS.
    - The ZKP(s2) and ZKP(s3) in order to verify the correctness of the partial decryption.
11. **Issuing the voting receipt:** after all the verifications, the RCG digitally signs the voting receipt, sends stores it in the RCG database and sends it to the VCS.
12. **Storing the vote in the Ballot Box:** the voting receipt is the confirmation of the correct submission of the vote for the VCS, which stores the encrypted vote received from the voter, updates the local Electoral Roll (eRoll) and forwards the voting receipt to her. The storage of the vote and update of the Electoral Roll must be an atomic operation.
13. **Confirmation of correct ballot casting:** when the voting receipt is received in the voter side, the Voting Client confirms to the voter that the ballot casting has been successful.
14. **Return Codes:** after issuing the voting receipt, the RCG calculates the Return Codes by partially decrypting the – already – partially decrypted voting options (E'(vote)).

    The Return Codes are searched in the RCG database in order to translate the Candidate Return Codes into Position Codes (pre-generated info). The Return Codes (parties) and the Position Codes are sent to a SMS gateway jointly with the voter phone number (extracted from the Ta).
15. **Submission of the Return Codes by SMS:** finally, the Return Codes are sent to the voter by SMS.

1.5.3 Election Tally Process

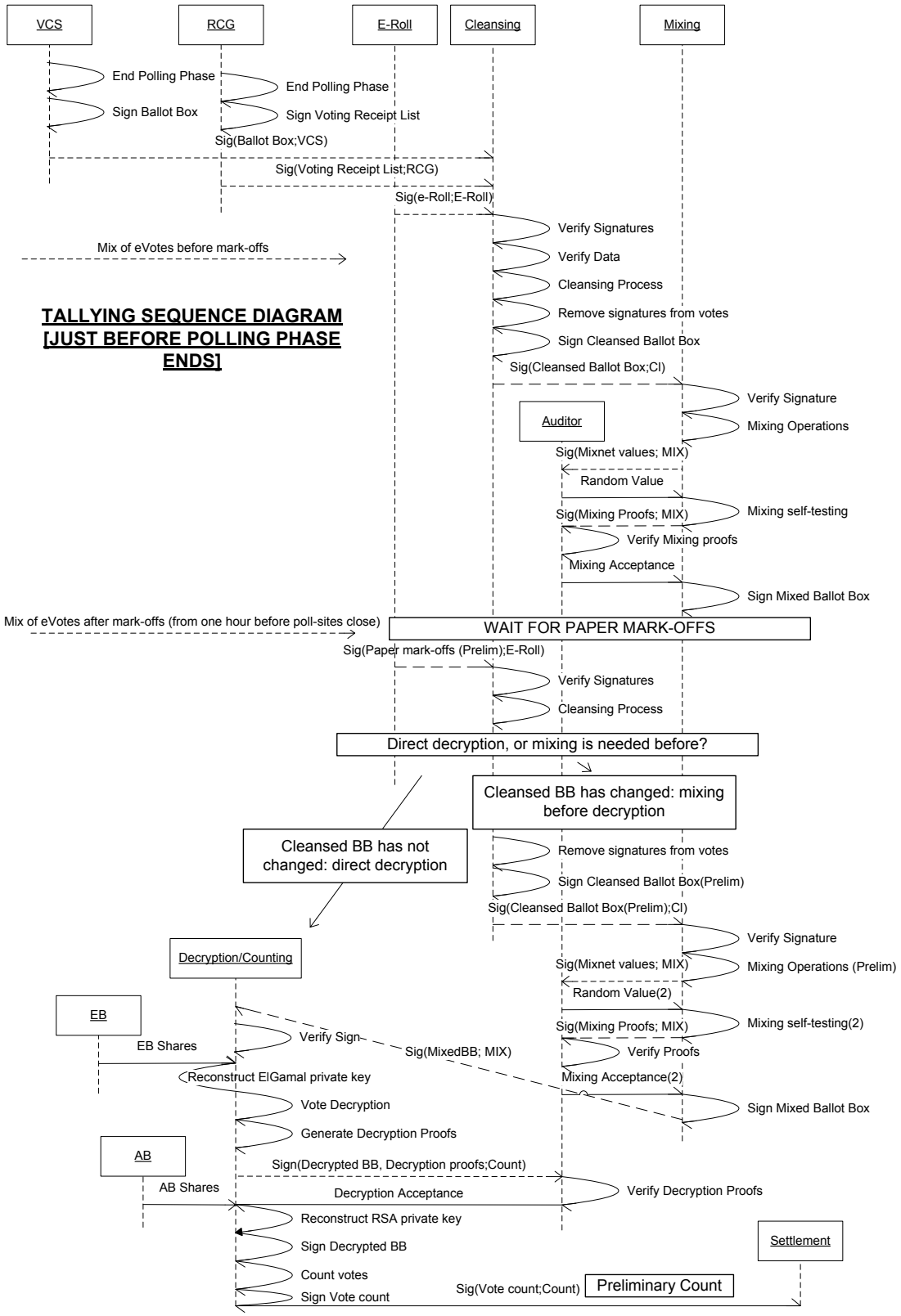
The following phases can be distinguished in the tallying process:

- **PHASE A:** *Preliminary Mix.* Cleansing and mixing of the electronic votes.
- **PHASE B:** *Preliminary Count.* First arrival of paper mark-offs half an hour or one hour before the voting phase ends. The cleansing is repeated (with all the electronic votes). In case the cleansed ballot boxes do not change (people who have voted in electronic, have not voted in paper) the mixed ballot boxes can be decrypted with the EB private key. Otherwise, the mixing needs to be repeated before decryption.
- **PHASES C-X:** *Final Count Iterations.* Periodic arrivals of more paper mark-offs as the polling places start to close (and the Admin. System receives the order of generating a new final count). The cleansing is repeated (with all the electronic votes and all the received paper mark-offs). In case the cleansed ballot boxes do not change (people who have voted in electronic, have not voted in paper) the mixed ballot boxes can be directly decrypted using an HSM. Otherwise, the mixing needs to be repeated before decryption.
- **PHASE Z:** *Last Final Count.* The cleansing is repeated (with all the electronic votes and all the paper mark-offs). In case the cleansed ballot boxes do not change, the mixed ballot boxes can be directly decrypted with the EB private key. Otherwise, the mixing needs to be repeated before decryption.

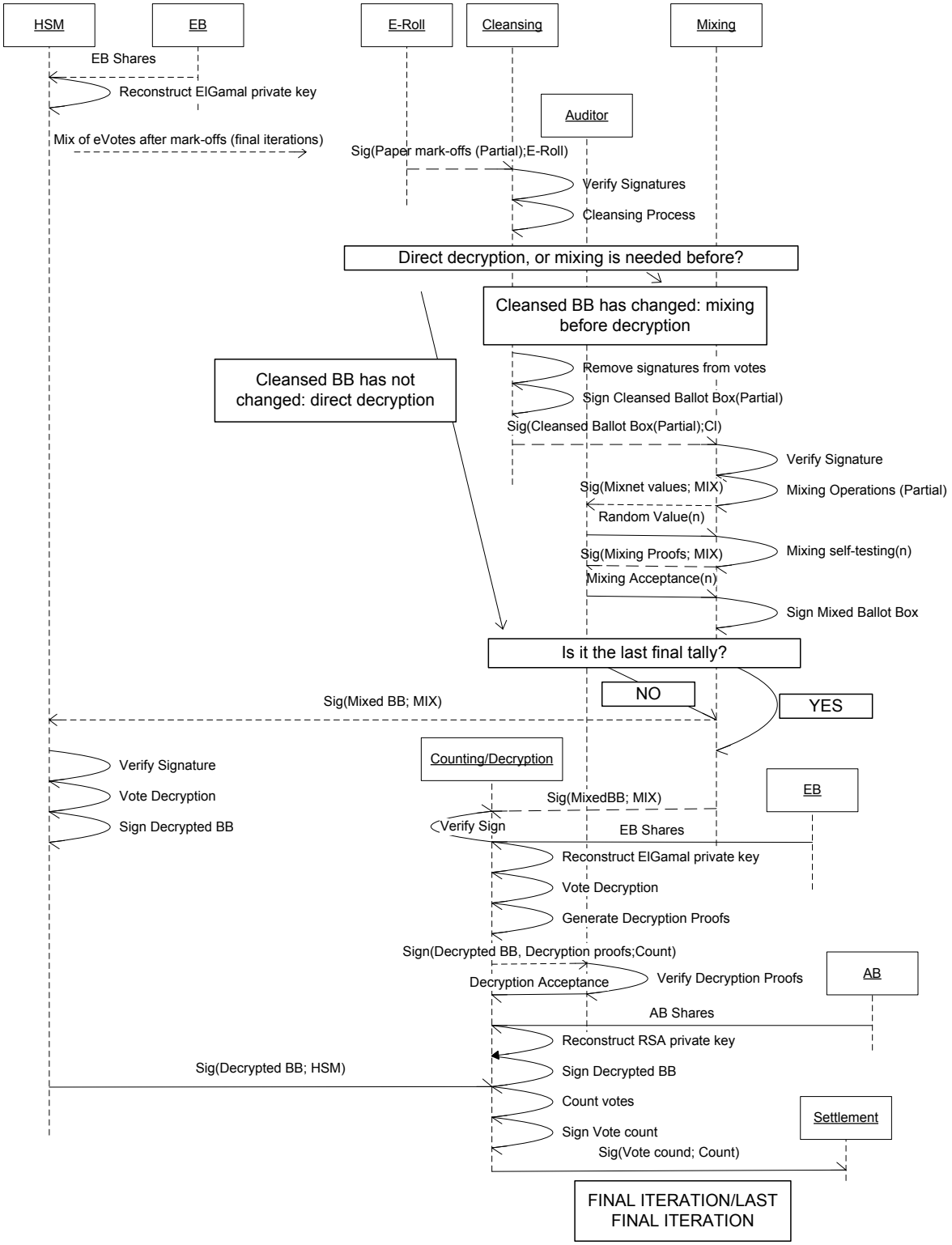
  All the eVotes are decrypted each time in phases C-Z.

1.5.3.1 Election Tally phase

The following sequence diagram depicts the interaction between the different modules during the election tally phase:



TALLYING SEQUENCE DIAGRAM [FOR FINAL DATA ITERATIONS]



1.5.3.2 Specific notation

Before explaining the steps of the election configuration phase, some specific notation must be explained:

- EB: Electoral Board.
- AB: Administration Board.
- Paper mark-offs or mark-offs: Electoral Roll marked with those voters who have voted on paper.
- HSM: Hardware Secure Module.

1.5.3.3 Steps of the election tally phase:

The steps presented in the sequence diagram are explained here with more detail.

PHASE A: *Preliminary Mix.*

End of the electronic polling phase

1. **Exporting the Ballot Box:** when the polling phase ends, the Ballot Box stored in the VCS is digitally signed and exported (offline) to the Cleansing Service.
2. **Exporting the list of Voting Receipts:** the list of Voting Receipts, stored in the RCG, is also digitally signed and exported (offline) to the Cleansing Service.
3. **Exporting the Electoral Roll:** the Electoral Roll containing the eligible voters, their authorised contests, and the corresponding digital certificates in case of assignments on-the-fly, is digitally signed by the Electoral Roll module and exported (offline) to the Cleansing Service.

Mix of eVotes before paper mark-offs have arrived

4. **Ballot Box Validation in the Cleansing Service:** after getting the Ballot Box, the list of Voting Receipts, and the Electoral Roll, the Cleansing Service performs some verifications to ensure the validity of the votes in the Ballot Box:

    - The digital signatures of the Ballot Box, the Electoral Roll and the list of Voting Receipts are validated.
    - The individual signatures of the encrypted votes, the issued Ta's, the TminID's and the Voting Receipts are validated. Also, the Schnorr signature of the votes is validated.
    - The information of the Ballot Box is compared to the information of the list of Voting Receipts:
        - Verify that each vote in the Ballot Box has its corresponding Voting Receipt.
        - Verify for each vote that the difference between the timestamps on Ta and TminID, and the timestamp on the corresponding Voting Receipt does not exceed the limit of time between authentication and voting.
        - Verify for each vote that the timestamps on Ta and TminID do not exceed the voting period.
        - Verify for each vote that the timestamp in the corresponding Voting Receipt does not exceed the time of the voting period plus some established margin.
        - Verify that the contest of a specific vote in the Ballot Box is in the list of authorised contests in the Ta, and that these authorised contests match those specified in the Electoral Roll for the voter who cast the vote.

- Verify for each vote that both the Ta and TminID identifiers are unique for the same contest.
- Verify for each voter that the SSN in the voter digital certificate is the same that in TminID (by using the list of on-the-fly assignments if it is the case)
- Verify that the length of the vote is the same than the maximum defined for those contests.

5. **Cleansing the votes:** after verifying that all the votes in the Ballot Box are valid (and rejecting those invalid), the Cleansing Service chooses, for each voter (who has voted), the vote with higher timestamps (in Ta, TminID, Voting Receipt) for the same contest. Logs are generated about the "discarded votes": number of erased votes, SSN/H(vote), reason.
6. **Generating Cleansed Ballot Box by district:** also, votes in the Cleansing Service are grouped by districts (in order to perform an independent shuffling for each one). Finally, the signatures of the voters are removed from the votes, the Cleansed Ballot Box is signed by the Cleansing Service, and it is exported (offline) to the Mixing Service.

   Note: since the mixing is performed independently by district, we will consider here that a Cleansed Ballot Box is formed by the votes of one specific district, and that the procedure is done for all the districts (several Cleansed Ballot Boxes).
7. **Mixing the votes:** the first node of the Mixnet receives the Cleansed Ballot Box and verifies its digital signature. After that, the votes are shuffled and re-encrypted across all the mix-nodes.
8. **Verifying the mixing process-commitment of mixing values:** after the mixing process, the inputs and outputs of each mix-node are digitally signed (by that mix-node), and delivered to the auditor external equipment or machine (off-line).
9. **Verifying the mixing process-challenge for the Mixnet:** the auditor then introduces a random value in the first mix-node, which is used to generate ZK Re-encryption Proofs. The other nodes in turn also generate ZK Re-encryption Proofs. After verifying in the same Mixnet that the proofs have been correctly computed, they are digitally signed by the respective mix-node and delivered to the auditor external equipment, where the digital signatures and the proofs are verified. After that, the mixing is considered valid.

PHASE B: *Preliminary Count.*

Mix of eVotes when receiving paper mark-offs one hour before poll-sites close

Note: the following steps are followed when the paper mark-offs in the poll-sites are collected one hour before closing.

10. **Importing paper mark-offs:** one hour before the polling phase for paper votes ends, the Electoral Roll where voters who have voted in paper in a specific poll-site are marked (that is, paper mark-offs) is digitally signed (by the E-Roll service) and exported to the Cleansing Service to start a preliminary count, where the digital signature is verified.
11. **Repeating the Cleansing:** in the Cleansing Service, the votes in the –already- Cleansed Ballot Box belonging to voters who are present in the mark-offs (that is, who have voted in paper), are discarded from the Cleansed Ballot Box. This process is done for all the Cleansed Ballot Boxes. Logs are generated about this "discarded votes": number of erased votes, SSN/H(vote), reason.

12. **Deciding if we repeat the Mixing or we decrypt:** since there is a possibility that voters who have voted electronically have not voted in paper, the new Cleansed Ballot Box could be equal to the old one (no changes). In this case, the process jumps to de decryption process (step 14). If the new Cleansed Ballot Box is different to the old one, the Mixing has to be repeated.
13. **Repeating the mixing:** the new Cleansed Ballot Box is digitally signed and exported to the Mixnet, where a new mixing is performed.

    Note: only the mixing of the Cleansed Ballot Boxes which have been updated after the reception of paper mark-offs must be repeated.

    Steps 7, 8 and 9 are repeated with the updated Cleansed Ballot Boxes.

    Logs are generated about these updated Cleansed Ballot Boxes: number of votes in the old Cleansed Ballot Box ←→ number of votes in the old Mixed Ballot Box, number of votes in the new Cleansed Ballot Box ←→number of votes in the new Mixed Ballot Box.
14. **Vote decryption:** the votes resulting from the new mixing (or from the old one if there were not changes in the Cleansed Ballot Box due to paper mark-offs) are ready to be decrypted. Therefore, the Mixed Ballot Box is digitally signed by the Mixing service and sent (offline) to the Decryption/Counting service. There, the members of the Electoral Board provide their shares to reconstruct the ElGamal private key (in this Decryption/Counting service), and all the votes are decrypted.
15. **Decryption process verification:** the decryption process generates ZK decryption proofs. *[After that, the reconstructed ElGamal private key should be erased from the decryption service]*. The ZK decryption proofs and the Decrypted Ballot Box are digitally signed and collected (offline) by the Auditor's machine, where the proofs are verified.
16. **Signing the Decrypted Ballot Box:** after that verification, the Administration Board members provide their shares to reconstruct the RSA private key in the decryption/counting service and sign the Decrypted Ballot Box. *[After that, the reconstructed RSA private key should be erased from the decryption service]*.
17. **Counting the decrypted votes:** the decryption/counting service extracts the votes for each candidate by:
    - Factorizing each vote to extract the prime numbers which compose it.
    - Looking at the equivalence between prime numbers and candidates from a pre- generated list.
18. **Exporting the vote counts:** after obtaining the vote count per candidate (in a specific contest) values, these counts are digitally signed by using the RSA Decryption/Counting Service private key, and they are exported to the Settlement module, which is allocated in the Administration module.

Just to be done once, after the first mark-offs are collected

19. **Electoral Board private key reconstruction in HSM:** after the preliminary count has been done, the Electoral Board members provide their shares to an HSM, where the EB ElGamal private key is reconstructed and securely stored.

PHASES 3-X: *Iterations of Final Counts.*

Mix and count of eVotes every time paper mark-offs arrive for a final count iteration (e.g, each time a poll-site closes

Note: the following steps are followed when more paper mark-offs arrive. The process is repeated for each data arrival.

20. **Importing paper mark-offs:** when more paper mark-offs are available, they are digitally signed (by the E-Roll service) and exported to the Cleansing Service, where the digital signature is verified.
21. **Repeating the Cleansing:** in the Cleansing Service, the votes in the –already- Cleansed Ballot Box belonging to voters who are present in the mark-offs (that is, who have voted in paper), are discarded from the Cleansed Ballot Box. Logs are generated about these "discarded votes": number of erased votes, SSN/H(vote), reason.
22. **Deciding if we repeat the Mixing or we decrypt:** since there is a possibility that voters who have voted electronically have not voted in paper, the new Cleansed Ballot Box could be equal to the old one (no changes). In this case, the process jumps to the decryption process (step 24). If the new Cleansed Ballot Box is different to the old one, the Mixing has to be repeated (with the new Cleansed Ballot Box).
23. **Repeating the mixing:** the new Cleansed Ballot Box is digitally signed and exported to the Mixnet, where a new mixing is performed.

    Note: only the mixing of the Cleansed Ballot Boxes which have been updated after the reception of paper mark-offs must be repeated.

    Steps 7, 8 and 9 are repeated with the updated Cleansed Ballot Boxes.

    Logs are generated about these updated Cleansed Ballot Boxes: number of votes in the old Cleansed Ballot Box ←→ number of votes in the old Mixed Ballot Box; number of votes in the new Cleansed Ballot Box ←→ number of votes in the new Mixed Ballot Box.

24. **Vote decryption:** the votes resulting from the new mixing of the votes (or from the old mixing if there were not changes in the Cleansed Ballot Box due to paper mark-offs) are ready to be decrypted. Therefore, the Mixed Ballot Box is digitally signed by the Mixing service and sent (offline) to the HSM where the Electoral Board and Administration Board private keys are stored. There, the votes are decrypted using the EB private key, and the Decrypted Ballot Box is digitally signed using the HSM signing private key.

    All the electronic votes are decrypted in each iteration, not only those from the 'updated' Ballot Boxes.
25. **Counting the decrypted votes:** after the mixed votes are decrypted and the result is signed in the HSM, the Decrypted BB is sent to the Counting/Decryption module where the votes for each candidate are extracted by:
    - Factorizing each vote to extract the prime numbers which compose it.
    - Looking at the equivalence between prime numbers and candidates from a pre- generated list.
26. **Exporting the vote counts:** after obtaining the vote count per candidate (in a specific contest) values, these counts are digitally signed by using the RSA Decryption/Counting Service private key, and they are exported to the Settlement module, which is allocated in the Administration module.

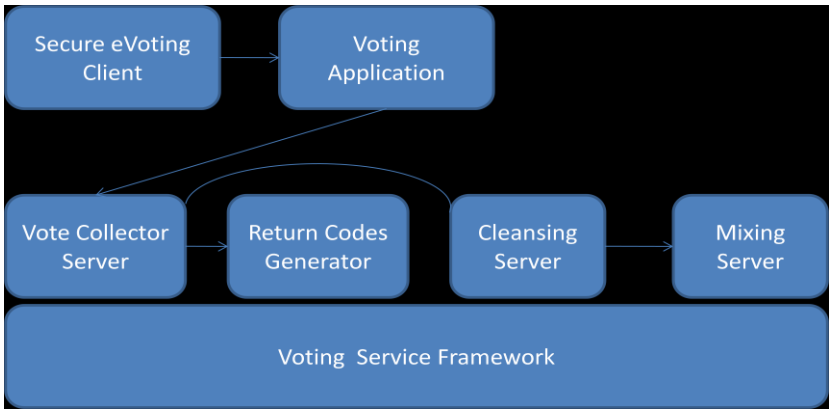
PHASE Z: *Last Final Count.*

Mix and count of eVotes when all the paper mark-offs have arrived

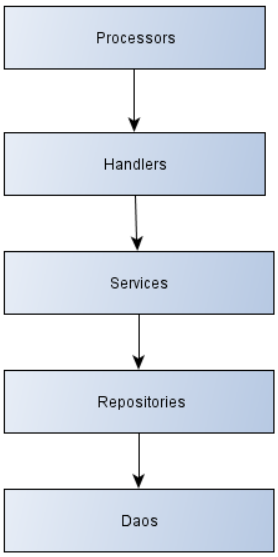
27. **Vote decryption:** when all the final paper mark-offs have arrived (this is the last iteration of the process), we decrypt again **all** the Cleansed and Mixed Ballot Boxes with verifiability properties: the Mixed Ballot Boxes are digitally signed by the Mixing service and sent (offline) to the Decryption/Counting service. There, the members of the Electoral Board provide their shares to reconstruct the ElGamal private key (in this Decryption/Counting service), and the votes are decrypted generating ZK decryption proofs.

After that, steps 15-18 are repeated with all the Ballot Boxes. This is the ***final verifiable count.***

1.6 Implementation overview



All server components in the subsystem will be web services that will use a common framework called the Voting Service Framework. It will provide a set of services for all the different servers to use.



Processors: This is the entry point of communication with the Voting Service Framework. They manage all requests made to the server within an specific group specified with the url's path of the web service invoked, for example, the "/voting" path. The system will have several processors like VotingProcessor for the requests related to the voting process, AdminProcessor for election configuration management requests, and others.

Handlers: A processor handles a group of requests, while each handler manages an specific request. For instance, the system will have VoteMessageHandler, LoginMessageHandler, etc. They have access to communication beans, deserializing requests, performing tasks and composing web service response.

Services: This layer is in charge of business logic, so for example the system will include BallotBoxService for all ballots related operations, ElectionConfigurationService for election configuration tasks, and so on. They also have access to entity objects, the ones that are persisted.

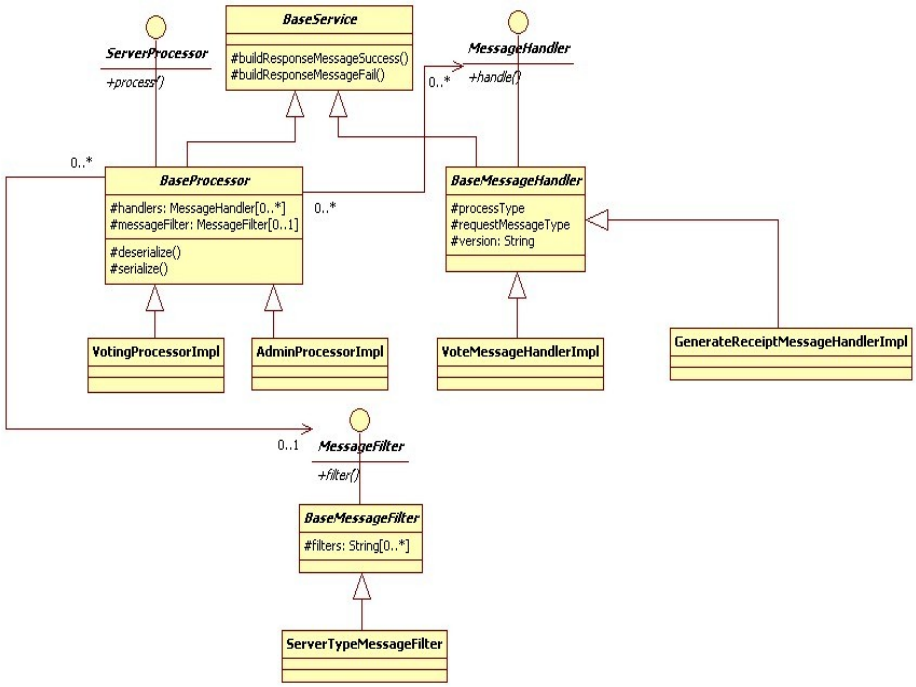
Repositories: Repositories are in charge to concentrate database access operations. They communicate with persistence layer through generic DAOs. For instance the system will have BallotBoxRepository, RollRepository, etc.

DAOs: This layer consists of generic DAOs, mainly one that provides access DAO operations using Hibernate and other one providing database access trough JDBC.

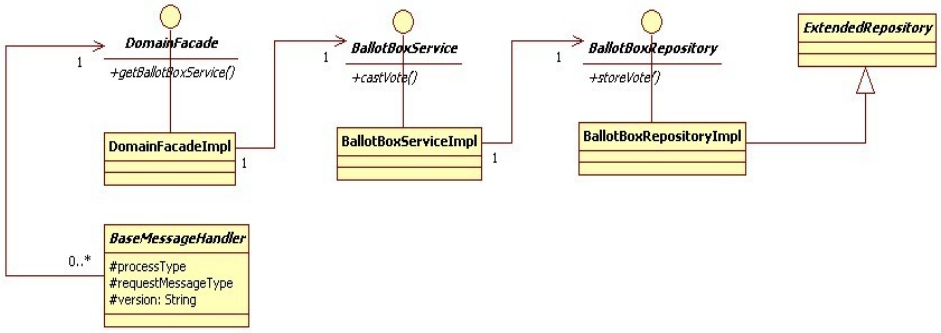
The entry point for web services to communicate with the server is managed by the processors. They receive every request and forwards them to the corresponding handler base on a filter applied according to the message request type and the server type that is running (as the framework is prepared to run as a VCS instance, a RCG server instance or other depending on its configuration).

The handlers are in charge of deserializing each request to obtain the specific request communication bean (the request information) and call service layer to perform a task and after that, compose the web service response with the appropriate communication bean.

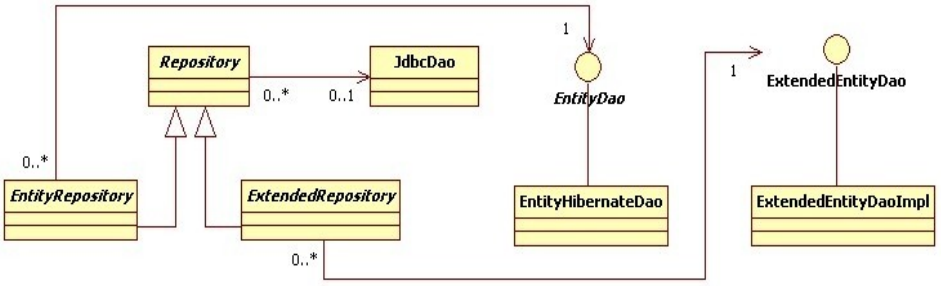
As part of the service layer, there is one single facade to access all services available for the application from the handlers' layer. The services perform specific tasks relative to business logic and call the repository layer for DAO operations. Finally, each repository has access to generic DAOs for persistence operations.



Processor and handler layers



Service layer



DAO layer

1.7 Data model

The data model is quite simple and contains a limited number of tables. The most significant ones are the ballot box and the receipts table.