Trusted Civitas: Client Trust in CIVITAS Electronic Voting Protocol

João Mendes[1] and Pedro Adão[1,2⋆]

[1] Instituto Superior Técnico, Technical University of Lisbon
[2] SQIG-Instituto de Telecomunicações
jmbsm@ist.utl.pt pedro.adao@ist.utl.pt

Abstract. Electronic voting promises the possibility of a convenient, efficient and safe way to capture and count votes in an election. Recently, Clarkson et al. [4] proposed Civitas, a voting scheme that ensures both security and privacy for the voter. In spite of its cryptographic robustness, Civitas is not yet ready to be used in the real world as it still has significant vulnerabilities, namely regarding the trust in the voting client. In this paper we propose a solution based in the usage of smart cards that is resilient against compromise of the voting client, maintains all the security and privacy properties of Civitas, and allows the user to have a confirmation of his cast vote at the expense of having another trusted, but simple, central authority. With this proposal, we do not require the existence of code cards that need to be issued for each election, contrary to what is common for solving this problem.
Keywords: E-Voting, Privacy, Coercion-Resistance, Trust Voting Client

1 Introduction

Electronic voting protocols are special cases of security protocols that are run between voters and administrators. Security protocols have been studied for a while but voting protocols present themselves as a special case as the intended security goals, verifiability and privacy, may look somewhat contradictory. The current state of secure electronic voting is far from perfect as major commercial electronic voting systems fail to offer strong privacy guarantees, a fact well known to the community.

Traditional Paper Voting All of us are very used to paper voting where a ballot is a piece of paper used to record voters' choices. Each voter uses one ballot, ballots are not shared, therefore each ballot is unique. In simple elections, a ballot may be a simple scrap paper in which each voter writes or selects his candidate, but in real world usage, *e.g.*, governmental elections, pre-printed ballots are used to protect the privacy of the voters. Humans have a profound affinity for that which they can see and touch. This results in a deep reverence for the printed word, whether it is true or false, and explains the comfort people derive from paper receipts.

⋆ Partially supported by FCT project ComFormCrypt PTDC/EIA-CCO/113033/2009.

However, there are various problems inherent to paper-based elections. Logistics, cost and resources involved, the time needed for all phases of the electoral process (printing votes, tally votes), etc, are non-trivial and electronic voting presents itself as a solution for many such problems.

Types of Electronic Voting Three approaches to the problem of electronic voting have been proposed so far [11]:

- *Poll-site voting* commonly seen as direct recording electronic (DRE), are special machines that are present in polling stations and have dedicated hardware and software. Voters cast their votes interacting with such a machine, and in some cases get a receipt for verification;
- *Kiosk voting* voting takes place through publicly available terminals;
- *Voting via Internet* performed by a client-server application, run by voter's PC, mobile phone or PDA, and on the server side, by trusted authorities. Neither the terminal not the environment can be controlled [12].

Internet voting systems are obviously the most appealing for several reasons. One strong reason is convenience; it allows a voter to vote from anywhere, which might be seen as a way to decrease abstention rates. Also, people are getting used to perform sensitive operations such as shopping and home banking from their home computers. However, Internet voting systems face several problems that prevent their widespread use today, that can be broadly divided in three main classes.
The first class deals with the problems that are specific to voting protocols. These problems derive from the assumptions of the protocols about the execution environment. The second class includes those difficulties that may be created by specific attacks against a voting protocol or a running election. Such attacks may try to get some useful outcome, by subverting the voting protocol, or simply ruin an election using *Denial of Service* attacks against the participating machines or applications. Another possible attack is the coercion of voters, which can happen if voters can vote anywhere without supervision of electoral committees. Finally, there is another class that includes security and fault tolerance issues inherited from the current Internet architecture. In this paper we address the problems present in the first two classes. Building on top of Civitas, that ensures both security and privacy of electronic voting, we propose a solution that does not require trust in the voting client.

Security Properties Electronic Voting System should fulfill some properties, in order to be secure and therefore usable in real-world [5,8]. Most of these are inspired in paper-based voting and derive from there:

- *Eligibility*: only legitimate voters can vote;
- *Fairness*: no early results can be obtained as they could influence the remaining voters;
- *Individual Verifiability*: a voter can verify that her vote was really counted (at some point at the elections);
- *Universal Verifiability*: the published outcome of the tally can be verified to be the sum of all the cast votes;
- *Vote-Privacy*: the fact that a particular voter voted in a particular way is not revealed to anyone;

- *Receipt-Freeness*: a voter does not gain any information, a receipt, that can be used to prove to a coercer the way she voted;
- *Coercion-Resistance*: a voter cannot cooperate with a coercer to prove to him that she voted in a certain way.

Coercion-resistance is the strongest of the three privacy properties and, naturally, should be the intended privacy goal of an e-voting scheme.

Our Work We propose a solution based in the usage of smart cards and an external reader that is resilient against compromise of the voting client, maintains all the security and privacy properties of Civitas such as verifiability and coercion-resistance, and allows the user to have a confirmation of his cast vote. This is done at the cost of those two simple hardware devices and the need for one extra trusted party.

2 Related Work

End-to-end auditable or end-to-end voter verifiable (E2E) systems are voting systems with stringent integrity properties and strong tamper-resistance, that allows all voting phases to be verified at some point in the election period.

Prêt à Voter Prêt à Voter [13] is a voting system that provides verification of the ballot. It allows voters to verify that their votes have been included in the count while ensuring their vote remains secret. Voters mark their selection on a paper ballot form in the usual way against the candidates available. The key novelty is that the candidates are listed in a random order, which varies from ballot to ballot. After the elections' closing, the system publishes on a public web bulletin board the receipts of all the votes it has accepted. Each stage that the votes go through in the system can be independently verified. Prêt à Voter offers a weak form of coercion resistance, if voting is supervised. In remote settings, it offers no coercion resistance. The adversary, by observing the voter during voting, will learn what vote was cast.

SureVote SureVote [2] is a commercial, special-hardware enhancement of the mixnet approach by Chaum which incorporates a "visual crypto" voter-verifiable component. SureVote generates secret vote and reply codes for each candidate and for each voter. The codes are delivered to the voters prior to the election day. On election day the voter sends the vote code of her/his favorite candidate through the voting channel, e.g. Internet. At server side, the reply code is computed by a set of trustees and sent to the voter that confirms it – in this way it is verified that there was no vote modification. If there is at least one corrupted trustee, SureVote does not guarantee that in the counting phase the vote code is translated to the right candidate.

3 Civitas[3]

Our solution is based in Civitas [4] and we briefly describe it in here. We refer the reader to [4] for full details and discussion. Civitas implements

[3] Part of this Section is extracted from [4].

a voting scheme proposed by Juels, Catalano, and Jakobsson [7]. There are seven different entities performing different roles in CIVITAS:

- The *Supervisor* administers the election. It specifies the ballot design, the election authorities, and is responsible for opening and closing the election;
- The *Registrar* decides eligible voters for the election;
- The *Registration Tellers* generate the credentials that allow voters to cast their votes;
- *Voters* are the individuals that participate in the election; they register themselves, get their credentials, and cast their votes;
- *Ballot Boxes* are instances of an insert-only log service. Are used by voters to cast their votes, and report their results to the Tabulation Tellers once the election is closed;
- *Tabulation Tellers* tally votes after the closing of the election and compute the final outcome;
- *Bulletin Board* (BB) is a write-only service, readable by everyone, where the electoral authorities post all the necessary information for verifiability of the election.

Setup phase (1) The *supervisor* starts the election posting in the *bulletin board* the ballot design, and the public key of every electoral authority. These keys are used for authentication and encryption of exchanged messages; (2) the *registrar* posts the *electoral roll*, containing identifiers for all authorized voters, along with their *registration* and *designation* (public) keys; (3) *tabulation tellers* generate a public key K_{TT} for a distributed El Gamal encryption scheme and post it on the *bulletin board*. Decryption of messages encrypted under this key requires the cooperation of all tabulation tellers; and (4) *registration tellers* generate the *voting credentials* that are used to (anonymously) cast votes.
Each registration teller i authenticates a voter using the voter's registration key. Then the teller and the voter run the registration protocol, using the voter's designation key, to generate the voter's private credential share s_i and posts on the bulletin board the public share of the credential, $S_i = \mathsf{Enc}_{K_{TT}}(s_i)$. Private credential $s = \prod_i s_i$ can only be forged if all registration tellers collude. Due to the homomorphic property of the encryption scheme, the public credential of each voter can be publicly computed as the product of all her public shares as

$$S = \prod_i S_i = \prod_i \mathsf{Enc}_{K_{TT}}(s_i) = \mathsf{Enc}_{K_{TT}}(\prod_i s_i).$$

Voting phase After obtaining the private credentials, the voter is able to cast her vote whenever she wants. The voter submits the tuple

$$\mathsf{Vote}(s, v) = \langle \mathsf{Enc}_{K_{TT}}(s), \mathsf{Enc}_{K_{TT}}(v), P_w, P_k \rangle$$

to some or all of the ballot boxes containing her private credential s, her *choice* of candidate v, along with a proof P_w that the vote is well-formed, and a proof P_k that the voter knows both s and v. The presence of proof P_w allows the verification of the correct construction of the vote

without the need to open it, and the proof P_k prevents an adversary from resubmitting old credentials $\mathsf{Enc}_{K_{TT}}(s)$ with new votes $\mathsf{Enc}_{K_{TT}}(v')$.

Tabulation phase To compute the outcome of an election, all tabulation tellers have to cooperate: (1) all tellers *retrieve* the public credentials of each voter from the bulletin board and the votes from each ballot box; (2) *verify that the proof* of well-formedness P_w is valid for each vote and discard the ones with invalid proofs; (3) *Votes with duplicate credentials are identified* performing *PET*[4] *and are eliminated* according to the revoting policy; (4) the lists of submitted votes and of authorized credentials in the bulletin board *are anonymized* using a *mix network* [3]; (5) the credentials in the anonymized list of submitted votes are compared with the anonymized list of authorized credentials using a *PET* and votes with *non-valid credentials are eliminated*; (6) final result of the election is computed decrypting the voter's choices, but not credentials.

3.1 Security Assumptions

To define the security of Civitas Clarkson et al. defined a set of assumptions about the trustworthiness of agents and system components.

- *Assumption* 1*: The adversary cannot simulate a voter during registration.* If the adversary may present himself as the voter at all times, then there is no way to distinguish the two.
- *Assumption* 2*: Each voter trusts at least one registration teller, and the channel from the voter to that teller is untappable.* To resist to coercion the voters need to be able to create fake credentials that look like real ones to the adversary and for that it is needed that at least one is not known to the adversary.
- *Assumption* 3*: Voters trust their voting client* as a corrupted client might reveal to the adversary the vote of the voter, or reveal all the private credentials defeating coercion resistance, or simply not send the vote to any ballot box causing the voter to abstain.
- *Assumption* 4*: The channels on which voters cast their votes are anonymous* otherwise the adversary could perform traffic analysis to know if a given voter has voted hence violating coercion resistance.
- *Assumption* 5*: At least one of each kind of authority is honest.* At least one of the ballot boxes to which a voter submits her vote is honest, otherwise could discard votes, and there exists at least one honest tabulation teller as if all were corrupted by decryption of the credentials and votes the adversary could violate coercion resistance.
- *Assumption* 6*: The Decision Diffie-Hellman and RSA assumptions hold, and SHA-256 implements a random oracle.*

3.2 Security Properties

Resisting Coercion To resist to coercion (and vote selling), voters have to be able to create fake credentials and provided them to the adversary

[4] *PET* stands for *Plaintext Equivalence Test* and is a ZK protocol that given c and c' reveals whether $\mathsf{Dec}(c) = \mathsf{Dec}(c')$ but nothing more about the plaintexts c and c'.

without him being able of distinguish them from valid ones. Then we have two possible scenarios: (1) if *the adversary asks the voter to vote in a particular candidate* then the voter casts her vote with a fake credential that will look good to the adversary but will be discarded later in the tabulation phase; (2) if *adversary asks the voter to sell her credential* the voter just gives him a fake credential that, for all observable purposes, is equal to a valid one.

Constructing a Fake Credential Fake credentials are created locally by voters using the voter's private designation key. This produces fake private shares s'_i that are indistinguishable from the real ones s_i to any adversary. These are combined to create the private share s' whose public credential is the original S.

Revoting Each voter may submit more than one vote. Votes made with fake credentials are eliminated but one still needs to decide what to do with multiple votes with same credentials. In these cases the supervisor may specify a policy on how to tally these votes. Either discards all or he needs to know which one to accept. Recall that votes are anonymized and mixed and so a proof of knowledge of the earlier votes has to be included in the vote that the voter wants to be counted.

Verifying an Election Civitas provides both *Universal and Individual Verifiability*. Universal verifiability is achieved requiring each tabulation teller to post a publicly verifiable proof that he is following the tallying protocol. Since everyone can check these proofs, honest tellers, and there is at least one by Assumption 5, stop once they detect any deviation from the tallying protocol. *Individual Verifiability* is achieved as voters can check if their vote is in the final tally of the election.

4 Proposed Solution

One major concern when deploying a system that allows remote voting is to be sure that the voting client does not behave erratically, and performs indeed the tasks that he is assigned to. Clarkson et al. [4] had this problem in mind when they created Civitas and solved it with a trust assumption, that we intend to eliminate now. Allowing a rogue voting client is the purpose of Joaquim et al. [6], that apply CodeVoting to have a *high level of trust in a voting client*, with reduced impact on the trust assumptions. One first try would be to integrate CodeVoting with Civitas but this is not obvious, hence the need to find an alternative solution. Other assumption in Civitas is that an *adversary cannot simulate a voter during registration*. This assumption may be simple to satisfy but we will remove it by assuming that registration is performed in-person and, a smart-card is given to the voter.
Our goal is to alter the least components of Civitas while removing the Assumption of trust in the voting client. Adding a smart-card may seem a big change that defeats the purpose of a remote voting system however, if one looks carefully, this does not disturb the process as this smart-card could be the national identification card already present in countries like

Estonia [9] and Portugal. This way, there is no extra hassle for the voters to obtain such card, hence the process is transparent.
This smart-card is an important piece of our architecture as one needs some computations to be performed correctly. Removing the trust on the voting client is not an easy task and several problems arouse. In the worst case, we may assume the client to behave erratically and not perform any of the intended tasks. Not even the expected computations of the protocol. Having the smart-card as a thrustworthy platform of computation helps us on this problem.

4.1 New Architecture

In our solution (Fig. 1) we add a new trusted entity to Civitas, the *VoteReplier*, that is responsible for replying to each voter with the correct reply-code, after reading the votes from the Ballot Boxes (Section 4.5). We also introduce two physical components on the voter's side that will allow us to vote without the need to trust the voting client: a smart-card, that can be the national identification card as one has in Portugal, and a smart-card reader with a keypad and a screen. Only these two components need to be certified. The keypad will be used to insert the intended vote and the screen to show the expected confirmation code. All the computations will be performed within the smart-card. The voting client is only needed to perform communication with the voting authorities and to show the confirmation code supplied by them. As one can see, these extra components in spite of trusted, are much easier to be verified than a whole operating system that is subject to worms and virus.

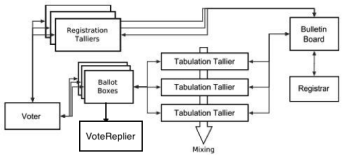

Fig. 1. New architecture based in Civitas.

The *Supervisor*, *Tabulation Tellers*, and *Bulletin Board* described in Section 3 are not changed in this new Architecture. *Voters* will perform the same tasks however votes will be cast differently (Section 4.4). The other three entities will have minor changes to support the existence of the *VoteReplier* and the issuing of the smart-card:

- *Registrar* performs in-person registration. It assigns a smart-card to each voter;
- *Registration Tellers* generate the credentials that are used to cast votes. The trusted credential, Assumption 2, will be placed inside the smart card issued by the *Registrar*;
- *Ballot Boxes* record the cast votes, and additionally report their contents to the *VoteReplier*.

4.2 Registration Phase

CIVITAS uses a remote agent *Registrar* in the *Setup Phase* to authorize voters. Authorized voters need to get their credentials from this agent and doing this remotely can be dangerous, as an adversary can impersonate a voter and steal his credential. The latter is what distinguishes a voter from an adversary and therefore it must be kept private. In spite of allowing this process to be done remotely, CIVITAS requires the impossibility of an adversary simulating a voter in registration (Assumption 1). In our solution, we change the remote agent *Registrar* by an *in-person Registrar* and this problem disappears.

The in-person *Registrar* provides the voter with a PIN code that allows the user to construct a valid or (intentionally) invalid credential, and a smart-card containing one of the private shares of the private credential. The other shares, obtained later from remote *Registration Tellers*, corrupted or not, can than be stored in and managed by the smart card solving also the problem of credential management. Publication of public credentials by the Tellers is done in the same way as in CIVITAS.

Much like in the Citizen's card enrollment, the voter must prove his identity, which is checked both by systems and humans. This reduces greatly the probability of identity theft. All relevant personal data and credentials are also stored inside the Citizen's card and, similarly to smart card's application, both its hardware and software are certified. Also, for the sake of transparency, its architecture is public. We consider that this change does not introduce too much disturbance in CIVITAS as it is a one time operation that the voter does not need to repeat.

4.3 New Hardware

Even using the smart-card to perform all computations, one could end up casting a vote different from the intended. If one uses the keyboard of the client, this could alter the introduced vote and provide a different one to the smart-card. Similarly, if we rely on the screen of the voting client, it could always present a default confirmation code that would obviously match with the (fake) confirmation code sent by the server.

In order to be sure that the inserted values are not corrupted or changed by any means, in particular an infected computer host, we require the user to have a certified smart-card reader. Its architecture should be public and both its *hardware* and *software* certified to ensure that no tampering is possible. We require it to have:

- a *Numeric keypad* to allow numeric inputs for voter's candidate choice and input PIN;
- a *Display* to show the input from voters and the expected reply-code;

This certified smart card reader would be distributed when voters obtain theirs Citizen card at the local authorities' office and should be *plug-and-play*. The assumptions on its security are that the numbers inserted in the keypad are indeed transmitted to the smart-card and that results shown in the display are the ones returned by the smart-card.

4.4 Voting Phase

Adding confirmation codes to the CIVITAS protocol allows the voter to confirm that his vote was registered correctly in the ballot box. However, it is still needed to convince the coercer, who could be right next to the voter, that the vote is valid and that it has arrived correctly to its destination. For that, we still need to maintain the credential faking ability from CIVITAS, to allow the voter to resist coercion, while at the same time use codes as a receipt confirmation. To cast a vote, the voter chooses a candidate and type in his choice in the keypad of the reader. The smart card asks for a PIN and computes the vote as

$$\mathsf{Enc}_{K_{VR}}(\mathsf{Vote}(s,v)) = \mathsf{Enc}_{K_{VR}}(\langle \mathsf{Enc}_{K_{TT}}(s), \mathsf{Enc}_{K_{TT}}(v), P_w, P_k \rangle)$$

that is exactly the same as in CIVITAS but with an extra outermost encryption with the *VoteReplier*'s key, and displays the value $\mathsf{Enc}_{K_{TT}}(v)$ in the display of the reader.
If the PIN is correctly inserted, then the vote is computed using the correct share s_i that is inside the smart-card and was given at the registration phase; otherwise, the vote will be cast with an incorrect share s' generated by the smart-card using the voter's private designation key. This uses the CIVITAS' ability to construct a fake credential, by making the smart card running a local algorithm to produce a fake credential in the vote. This is inline with what is required in CIVITAS to satisfy coercion-resistance.
Note that the vote will appear real and valid to everyone as in CIVITAS, except for the voter, who used the wrong PIN and consequently a wrong *private credential* on purpose to cheat on the coercer. Both the coercer and voter will receive the correct confirmation code from the *VoteReplier*, that matches the code that is visible on the smart card's display.

4.5 VoteReplier

As described in Section 3 the *Ballot Boxes* are insert-once and read-only entities in CIVITAS, that are distributed and eventually compromised. Since we do not want to change the *Ballot Box* properties from CIVITAS, we introduce a new trustworthy entity with the only function of reading the contents of the *Ballot Boxes*, recall from CIVITAS that the cast votes are still encrypted, and reply to the voting client with the confirmation code. Given a vote $\mathsf{Enc}_{K_{VR}}(\langle \mathsf{Enc}_{K_{TT}}(s), \mathsf{Enc}_{K_{TT}}(v), P_w, P_k \rangle)$ the *VoteReplier* decrypts the message and returns the value $\mathsf{Enc}_{K_{TT}}(v)$ that is the encrypted choice of the voter, as in CIVITAS, but in our case not known to the voting client. If this code matches the one in the display of the reader then the vote was correctly received by the *Ballot Box*. Otherwise it was tampered and the voter knows that he has to vote again. All CIVITAS assumptions regarding the ballot boxes remain intact.
The needed properties of the *VoteReplier* are that it does not reveal its private key K_{VR} to anyone and that it performs the decryption properly. Comparing with [6] no printed CodeCards are needed and every time a voter sends a vote, the reply-code is different. This saves time and

complexity in the voting process and is orthogonal to coercion resistance. Recall that for this process it is irrelevant if the credential in the vote is fake or not.

4.6 Tabulation

When the election closes, votes must be tallied by the tabulation tellers:

1. all *Ballot Boxes* commit received votes to the *VoteReplier*;
2. *Supervisor* posts a signed copy of all *Ballot Boxes* commitments;
3. all received votes are decrypted by the *VoteReplier* and committed to the *Bulletin Board*;
4. *Tabulation Tellers* proceed as in Section 3.

5 Security and Availability Analysis

We will show that all the extra information available to the adversary in our proposal could either be derived from Civitas or does not affect the security of the protocol. We also discuss the availability and implications of the new components and trusted parts of the architecture. We justify the necessity of those and why these restrictions are not problematic.

5.1 Correctness of the *VoteReplier*

It is easy to see that an adversary that does not know sK_{VR}, the secret part of the key, cannot forge a correct reply-code. Suppose that upon casting a correct vote

$$\mathsf{Enc}_{K_{VR}}(\mathsf{Vote}(s,v)) = \mathsf{Enc}_{K_{VR}}(\langle \mathsf{Enc}_{K_{TT}}(s), \mathsf{Enc}_{K_{TT}}(v), P_w, P_k \rangle),$$

there is an adversary A that is able to return the correct reply-code $\mathsf{Enc}_{K_{TT}}(v)$ without submitting the correct vote to the ballot boxes (if he submits it correctly, then the vote was correctly cast, and he obviously gets the correct reply-code from the *VoteReplier*). Then, one can use A to create an adversary A' that breaks CCA2 security of the encryption scheme. Adversary A' is defined as follows: given K_{VR},

1. construct $m_0 = \mathsf{Vote}(s, v_0)$ and $m_1 = \mathsf{Vote}(s, v_1)$,
2. submit it to the encryption oracle to obtain $c = \mathsf{Enc}_{K_{VR}}(\mathsf{Vote}(s, v_i))$,
3. give c to A to obtain $\mathsf{Enc}_{K_{TT}}(v_i)$,
4. return i.

Adversary A' breaks CCA2 security hence there is no adversary A that given $\mathsf{Enc}_{K_{VR}}(\mathsf{Vote}(s,v))$ returns the correct reply-code $\mathsf{Enc}_{K_{TT}}(v)$.

5.2 Security of the Trusted Civitas

With the changes introduced in our solution an adversary has access to the value sent to the ballot boxes $\mathsf{Enc}_{K_{VR}}(\mathsf{Vote}(s,v))$, and the value shown in the display of the trusted hardware $\mathsf{Enc}_{K_{TT}}(v)$.
These two pieces of information cannot violate the security of Trusted Civitas as both $\mathsf{Vote}(s,v)$ and $\mathsf{Enc}_{K_{TT}}(v)$ were already given to the adversary in Civitas without compromising security nor coercion resistance.

5.3 Availability of the System

An adversary can also perform attacks towards availability of the system. One possible attack towards availability is a DDoS attack against the *VoteReplier* as all votes have to pass through it. This attack is not a security attack and we may protect from it by having multiple instances as is the case for *BallotBoxes*. This requires these instances to have all the same key that would have to be kept secret.
Another big family of attacks are the ones where the reply-code is not correctly displayed to the user either because (i) it was not sent by the *VoteReplier*, or (ii) because it was maliciously altered by the voting client. For any of these cases there is no solution as we rely on the voting client to perform all the communications however, our solution at least provides a hint that something is wrong with the current voting client, information that was completely absent in Civitas. Both these cases may generate submission of repeated votes by the user in order to correct this error, however a user may decide to change voting clients if these occur quite often. Social attacks are also possible but out of the scope of this work.

5.4 Trust Assumptions of Our Architecture

We will now summarize the Trust Assumptions of our Proposal comparing with the ones of [4]. The new trust assumptions are marked as X'. The trust assumptions that remain unchanged keep the same numbering as in Section 3.1.

- ⊕ *Assumption* $1'$*: Registration is performed in-person.* This choice has a downside as the system is no longer fully remote. The upside is that credentials can be used in several elections.
- ⊖ Assumption 2 is dropped as with credentials are now inside the smart card. We need however these credentials to be private.
- ⊕ *Assumption* $2'$*: Credentials (and computations performed) in the smart card cannot be corrupted by any means* in a useful timeline.
- ⊖ Assumption 3 is also dropped. No need to trust the voting client.
- − *Assumption* 4*: The channels on which voters cast their votes are anonymous.* (already in Civitas) Only necessary to hide from which computers the votes are cast and this way avoid traffic analysis.
- ⊕ *Assumption* $5'$*: At least one of each kind of authority is honest.* (already in Civitas) The *VoteReplier* also needs to be trusted otherwise the reply-codes could be tampered.
- − *Assumption* 6*: The Decision Diffie-Hellman and RSA assumptions hold, and SHA-256 implements a random oracle.* (already in Civitas)

6 Conclusion

Automatic vote manipulation at client side is one of the biggest dangers that prevents the widespread of Internet voting. In this work we proposed a new architecture that extends Civitas in order to make it resilient against compromise of the voting client maintaining its coercion-resistance property. This new architecture adds three items to the original Civitas which make trust in client voting not necessary: (i) a *Smartcard*, to store credentials, compute and construct the vote; (ii) a certified

Smart Card Reader with keypad and display to use the smart card correctly and to display the expected confirmation code; and (iii) a trusted entity *VoteReplier* that returns the confirmation codes.
This solution is a trade-off between trusting all voting-clients, with all the diversity of voting clients/architectures/operating systems, and to trust the *VoteReplier*, a smart card reader, and the smart card itself that can be small certified components, simpler to verify, and easier to control. In fact, the main security assumption is in the smart-card itself. It performs all the computations and keeps the secret shares of the credentials.
We also introduce *in-person registration* that only needs to be performed once, hence not disturbing as other solutions resilient to compromise of voting client that need a new collection of code-cards for each election. This registration is anyways present in most European countries with the need for citizens to obtain some sort of identification card.
The security and privacy of our system is the same as in Civitas as all our changes are reduced to it with the benefit of our assumptions being much simpler. We consider this to be one more step towards a secure remote voting system. The needed changes should not be costly as national identity cards can be used as the smart-cards and the types of readers needed are simple and hence cheaper to produce in large quantities.

References

1. B. Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, pages 335–348. 2008.
2. D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.
3. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
4. M. Clarkson, S. Chong, and A. Myers. Civitas: Toward a secure voting system. In *IEEE Security and Privacy*, pages 354–368. 2008.
5. S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
6. R. Joaquim and C. Ribeiro. Codevoting: protecting against malicious vote manipulation at the voter's pc. In *Frontiers of Electronic Voting*, 2007.
7. A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *WPES*, pages 61–70. ACM, 2005.
8. S. Kremer, M. Ryan, and B. Smyth. Election verifiability in electronic voting protocols. In *ESORICS*, *LNCS*, pages 389–404. 2010.
9. E. Maaten. Towards remote e-voting: Estonian case. In *Electronic Voting in Europe*, volume 47 of *LNI*, pages 83–100. GI, 2004.
10. J. Mendes. Client trust in Civitas electronic voting protocol. TechReport, IST, Lisboa, 2010. MsC Thesis.
11. L. Nitschke. Secure remote voting using paper ballots. *CoRR*, abs/0804.2349, 2008.
12. R. Oppliger. How to address the secure platform problem for remote internet voting. *Proceedings of the SIS'02*. pages 153–173, 2002.
13. P. Ryan and S. Schneider. Prêt à voter with re-encryption mixes. In *ESORICS*, volume 4189 of *LNCS*, pages 313–326. Springer, 2006.