A different approach to code-voting
Achieving coercion-resistance and individual verifiability without the assumption of a trusted channel

Iñigo Querejeta-Azurmendi

Minsait, Indra
Universidad Carlos III de Madrid

October 23, 2017
Overview

- Project description
- Open problems
- Filtering stage
- Code-voting approach
- Project Objectives
**Project aim**

Present an **easy-to-use**, coercion-resistant and E2E verifiable internet voting protocol.

**Project motivation**

Give more guarantees than postal voting. Interest of developing countries to have an i-voting scheme where the only requirement is to have a pc/smartphone/tablet with internet connection.
Introduction

Project aim
Present an **easy-to-use**, coercion-resistant and E2E verifiable internet voting protocol.

Project motivation
Give more guarantees than postal voting.
Interest of developing countries to have an i-voting scheme where the only requirement is to have a pc/smartphone/tablet with internet connection.
Coercion-resistance ⇐⇒ E2E verifiability ⇐⇒ User friendly

Coercion-resistance
Offering the possibility to re-vote

E2E verifiability
Code-voting

I. Querejeta-Azurmendi
A different approach to code-voting
Overview

Coercion-resistance $\iff$ E2E verifiability $\iff$ User friendly

Coercion-resistance
Offering the possibility to re-vote

E2E verifiability

Code-voting
Overview

Coercion-resistance $\iff$ E2E verifiability $\iff$ User friendly

Coercion-resistance

Offering the possibility to re-vote

E2E verifiability

Code-voting
Re-voting

- Votes must be filtered prior to tally
- *Deniable voting* must be preserved
- *Universal verifiability* must be maintained in the filtering stage.

Actual solutions compare credentials in a provable hidden way.

Code-voting

- Highly used in e-voting
- Satisfies *receipt freeness*, but *coercion-resistance*?
- Vote-code sheets must be sent to voters (sms/post-mail/e-mail)

Usual assumption of a trusted channel
Contributions

Re-voting

- Votes must be filtered prior to tally
- *Deniable voting* must be preserved
- *Universal verifiability* must be maintained in the filtering stage.

Actual solutions compare credentials in a provable hidden way.

Code-voting

- Highly used in e-voting
- Satisfies *receipt freeness*, but *coercion-resistance*?
- Vote-code sheets must be sent to voters (sms/post-mail/e-mail)

Usual assumption of a trusted channel
Filtering has to remain private while maintaining Universal Verifiability

\[
\begin{array}{ccc}
E(id)_1 & E(vote_1) & \gamma_1 \\
E(id)_2 & E(vote_2) & \gamma_2 \\
E(id)_3 & E(vote_3) & \gamma_3 \\
E(id)_4 & E(vote_4) & \gamma_4 \\
E(id)_5 & E(vote_5) & \gamma_5 \\
\end{array}
\]

A usual solution is to blindly compare each of the credentials \([1, 2, 3, 4]\)

\[PET(E(id)_i, E(id)_j)\]

for each pair of encrypted identifiers \(E(id)_i, E(id)_j\).

In a distributed manner, resulting in high complexity.
Filtering stage

Filtering has to remain private while maintaining Universal Verifiability

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$E(id)_1$</td>
<td>$E(vote_1)$</td>
<td>$\gamma_1$</td>
</tr>
<tr>
<td>$E(id)_2$</td>
<td>$E(vote_2)$</td>
<td>$\gamma_2$</td>
</tr>
<tr>
<td>$E(id)_3$</td>
<td>$E(vote_3)$</td>
<td>$\gamma_3$</td>
</tr>
<tr>
<td>$E(id)_4$</td>
<td>$E(vote_4)$</td>
<td>$\gamma_4$</td>
</tr>
<tr>
<td>$E(id)_5$</td>
<td>$E(vote_5)$</td>
<td>$\gamma_5$</td>
</tr>
</tbody>
</table>

A usual solution is to blindly compare each of the credentials $[1, 2, 3, 4]$ 

$$PET(E(id)_i, E(id)_j)$$ 

for each pair of encrypted identifiers $E(id)_i, E(id)_j$.

In a distributed manner, resulting in high complexity.
Filtering stage

We use a column counter in the Web Bulletin Board (WBB), and we anonymously assign non-identifiable (but unique per voter) pseudonyms.

\[ \text{Cast: } = [\text{Enc}_{\text{ASK}}(\text{pseud}), \text{Enc}(\text{vote}), \gamma]_i \]

where the \text{pseud} is encrypted with an Assembling Server Key (ASK).

It is then stored in the WBB

<table>
<thead>
<tr>
<th>counter_1</th>
<th>$E_{\text{ASK}}(001)_1$</th>
<th>$E(\text{vote}_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter_2</td>
<td>$E_{\text{ASK}}(003)_2$</td>
<td>$E(\text{vote}_2)$</td>
</tr>
<tr>
<td>counter_3</td>
<td>$E_{\text{ASK}}(001)_3$</td>
<td>$E(\text{vote}_3)$</td>
</tr>
<tr>
<td>counter_4</td>
<td>$E_{\text{ASK}}(002)_4$</td>
<td>$E(\text{vote}_4)$</td>
</tr>
<tr>
<td>counter_5</td>
<td>$E_{\text{ASK}}(001)_5$</td>
<td>$E(\text{vote}_5)$</td>
</tr>
</tbody>
</table>
Filtering stage

We use a column counter in the Web Bulletin Board (WBB), and we anonymously assign non-identifiable (but unique per voter) pseudonyms.

\[ \text{Cast: } = [\text{Enc}_{\text{ASK}}(\text{pseud}), \text{Enc}(\text{vote}), \gamma]; \]

where the *pseud* is encrypted with an Assembling Server Key (ASK).

It is then stored in the WBB

<table>
<thead>
<tr>
<th>counter</th>
<th>$E_{\text{ASK}}(001)$</th>
<th>$E(\text{vote})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter 1</td>
<td>$E_{\text{ASK}}(001)$</td>
<td>$E(\text{vote}_1)$</td>
</tr>
<tr>
<td>counter 2</td>
<td>$E_{\text{ASK}}(003)$</td>
<td>$E(\text{vote}_2)$</td>
</tr>
<tr>
<td>counter 3</td>
<td>$E_{\text{ASK}}(001)$</td>
<td>$E(\text{vote}_3)$</td>
</tr>
<tr>
<td>counter 4</td>
<td>$E_{\text{ASK}}(002)$</td>
<td>$E(\text{vote}_4)$</td>
</tr>
<tr>
<td>counter 5</td>
<td>$E_{\text{ASK}}(001)$</td>
<td>$E(\text{vote}_5)$</td>
</tr>
</tbody>
</table>
Filtering stage

We use a column counter in the Web Bulletin Board (WBB), and we anonymously assign non-identifiable (but unique per voter) pseudonyms.

\[ \text{Cast: } = [\text{Enc}_{\text{ASK}}(\text{pseud}), \text{Enc}(\text{vote}), \gamma]_i \]

where the \textit{pseud} is encrypted with an Assembling Server Key (ASK).

It is then stored in the WBB

<table>
<thead>
<tr>
<th>counter</th>
<th>\text{Enc}_{\text{ASK}}(\text{digit})</th>
<th>\text{Enc}(\text{vote})</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter_1</td>
<td>\text{ASK}(001)_1</td>
<td>\text{E}(\text{vote}_1)</td>
</tr>
<tr>
<td>counter_2</td>
<td>\text{ASK}(003)_2</td>
<td>\text{E}(\text{vote}_2)</td>
</tr>
<tr>
<td>counter_3</td>
<td>\text{ASK}(001)_3</td>
<td>\text{E}(\text{vote}_3)</td>
</tr>
<tr>
<td>counter_4</td>
<td>\text{ASK}(002)_4</td>
<td>\text{E}(\text{vote}_4)</td>
</tr>
<tr>
<td>counter_5</td>
<td>\text{ASK}(001)_5</td>
<td>\text{E}(\text{vote}_5)</td>
</tr>
</tbody>
</table>
Filtering stage

Election closes $\rightarrow$ encrypt counters (with the Tallying Server Key (TSK))

$$E_{TSK}(\text{counter})_i \mid E_{ASK}(\text{pseud})_i \mid E(\text{vote}_i)$$

and the tally is shuffled using a provable mix-net. 

$pseud$ is then decrypted

<table>
<thead>
<tr>
<th>$E_{TSK}(\text{counter})$</th>
<th>001</th>
<th>$E(\text{vote})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{TSK}(\text{counter})_3$</td>
<td>001</td>
<td>$E(\text{vote}_3)$</td>
</tr>
<tr>
<td>$E_{TSK}(\text{counter})_2$</td>
<td>003</td>
<td>$E(\text{vote}_2)$</td>
</tr>
<tr>
<td>$E_{TSK}(\text{counter})_5$</td>
<td>001</td>
<td>$E(\text{vote}_5)$</td>
</tr>
<tr>
<td>$E_{TSK}(\text{counter})_4$</td>
<td>002</td>
<td>$E(\text{vote}_4)$</td>
</tr>
<tr>
<td>$E_{TSK}(\text{counter})_1$</td>
<td>001</td>
<td>$E(\text{vote}_1)$</td>
</tr>
</tbody>
</table>

Now we have groups of votes cast by the same voter, and we proceed to filter all votes but the last.
Filtering stage

Tallying Server proves correctness of filtering

\[ \pi = \text{ZKP}\{(E(\text{counter}_i)_{TSK}) : E(\text{counter}_i)_{TSK} > E(\text{counter}_j)_{TSK} \text{ for } j \neq i \text{ and } \text{pseud}_i = \text{pseud}_j\} \]

This ZKP consists simply in a proof of correct decryption. We use what is known as the 'Millionaires Protocol’, introduced by Brandt [5].

Number of ZKP are \(\mathcal{O}(v \cdot vc)\) where \(v\) is the number of voters and \(vc\) is the maximum number of votes cast by a single voter.
Filtering stage

Tallying Server proves correctness of filtering

\[ \pi = \text{ZKP}\{(E(\text{counter}_i)^{\text{TSK}}) : E(\text{counter}_i)^{\text{TSK}} > E(\text{counter}_j)^{\text{TSK}} \text{ for } j \neq i \& \text{pseud}_i = \text{pseud}_j\} \]

This ZKP consists simply in a proof of correct decryption. We use what is known as the 'Millionaires Protocol', introduced by Brandt [5].

Number of ZKP are \( O(\nu \cdot \nu_c) \) where \( \nu \) is the number of voters and \( \nu_c \) is the maximum number of votes cast by a single voter.
Filtering stage

Tallying Server proves correctness of filtering

$$\pi = ZKP\{(E(counter_i)_{TSK}) : E(counter_i)_{TSK} > E(counter_j)_{TSK} \text{ for } j \neq i \& \text{pseud}_i = \text{pseud}_j\}$$

This ZKP consists simply in a proof of correct decryption. We use what is known as the 'Millionaires Protocol’, introduced by Brandt [5].

Number of ZKP are $O(v \cdot vc)$ where $v$ is the number of voters and $vc$ is the maximum number of votes cast by a single voter.
Future work,

- Preparation phase of the ZKP may increase complexity
- Study trust assumptions
- ‘1009’ attack
Can we find a code-voting approach with no trust in the transport channel, and that offers deniable voting?

Standard code-sheets [6, 7, 8]:

<table>
<thead>
<tr>
<th>Voting Code</th>
<th>Verif Code</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC₁</td>
<td>VeC₁</td>
<td>Bradbury</td>
</tr>
<tr>
<td>VC₂</td>
<td>VeC₂</td>
<td>Orwell</td>
</tr>
<tr>
<td>VC₃</td>
<td>VeC₃</td>
<td>Hemingway</td>
</tr>
<tr>
<td>VC₄</td>
<td>VeC₄</td>
<td>Hesse</td>
</tr>
</tbody>
</table>

ballot_id
Can we find a code-voting approach with no trust in the transport channel, and that offers deniable voting?

Standard code-sheets [6, 7, 8]:

<table>
<thead>
<tr>
<th>Voting Code</th>
<th>Verif Code</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC₁</td>
<td>VeC₁</td>
<td>Bradbury</td>
</tr>
<tr>
<td>VC₂</td>
<td>VeC₂</td>
<td>Orwell</td>
</tr>
<tr>
<td>VC₃</td>
<td>VeC₃</td>
<td>Hemingway</td>
</tr>
<tr>
<td>VC₄</td>
<td>VeC₄</td>
<td>Hesse</td>
</tr>
<tr>
<td>ballot_id</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Can we find a code-voting approach with no trust in the transport channel, and that offers deniable voting?

Idea for the project:

<table>
<thead>
<tr>
<th>Voting Code</th>
<th>Verif Code</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>$VC_1$</td>
<td>$VeC_1$</td>
<td>Bradbury</td>
</tr>
<tr>
<td>$VC_2$</td>
<td>$VeC_2$</td>
<td>Orwell</td>
</tr>
<tr>
<td>$VC_3$</td>
<td>$VeC_3$</td>
<td>Hemingway</td>
</tr>
<tr>
<td>$VC_4$</td>
<td>$VeC_4$</td>
<td>Hesse</td>
</tr>
<tr>
<td>ballot_id</td>
<td></td>
<td>Posted WBB</td>
</tr>
</tbody>
</table>
Voter generates permutation matrix (e.g.: (1342))

\[ VC_1 \]
\[ VC_2 \]
\[ VC_3 \]
\[ VC_4 \]
Code-voting

Voter generates permutation matrix (e.g.: (1342))

<table>
<thead>
<tr>
<th>VC₁</th>
<th>VC₂(Brad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC₂</td>
<td>VC₄(Orw)</td>
</tr>
<tr>
<td>VC₃</td>
<td>VC₁(Hem)</td>
</tr>
<tr>
<td>VC₄</td>
<td>VC₃(Hess)</td>
</tr>
</tbody>
</table>
Voter generates permutation matrix (e.g.: (1342))

VC₁ → VC₂(Brad)
VC₂ → VC₄(Orw)
VC₃ → VC₁(Hem)
VC₄ → VC₃(Hess)
Voter generates permutation matrix (e.g.: (1342))

\[
\begin{align*}
& VC_1 \\
& VC_2 \\
& VC_3 \\
& VC_4
\end{align*}
\]

\[
\begin{align*}
& VC_2(\text{Brad}) \\
& VC_4(\text{Orw}) \\
& VC_1(\text{Hem}) \\
& VC_3(\text{Hess})
\end{align*}
\]

\[
\begin{align*}
& VeC_1 \\
& VeC_2 \\
& VeC_3 \\
& VeC_4
\end{align*}
\]
To cast a vote, a voter will send

\[ [VC_i, E(PemMat)_{VK}, ballot\_id, \gamma] \]

Then, the server, access the information of \textit{ballot\_id} and calculates a vector, \( VV_i \), with a one in the position of \( VC_i \)

\[ VV_i \times E(PemMat)_{VK} \]

And results in an encrypted vector with a one in the position of the candidate.

\[ (VV_i \times E(PemMat)_{VK}) \times E(PemMat)_{VK} \]

and it will result in a vector with a one in the position of the verification code.
To cast a vote, a voter will send

\[ [VC_i, E(PemMat)_{VK}, \text{ballot}_\text{id}, \gamma] \]

Then, the server, access the information of \( \text{ballot}_\text{id} \) and calculates a vector, \( VV_i \), with a one in the position of \( VC_i \)

\[ VV_i \times E(PemMat)_{VK} \]

And results in an encrypted vector with a one in the position of the candidate.

\[ (VV_i \times E(PemMat)_{VK}) \times E(PemMat)_{VK} \]

and it will result in a vector with a one in the position of the verification code.
To cast a vote, a voter will send

\[[VC_i, E(PemMat)_{VK}, ballot\_id, \gamma]\]

Then, the server, access the information of \textit{ballot\_id} and calculates a vector, \(VV_i\), with a one in the position of \(VC_i\)

\[VV_i \times E(PemMat)_{VK}\]

And results in an encrypted vector with a one in the position of the candidate.

\[(VV_i \times E(PemMat)_{VK}) \times E(PemMat)_{VK}\]

and it will result in a vector with a one in the position of the verification code.
Motivations to build such a working model:

- No trust in the voting device
- Coercion resistance
- No trust in the generation/sending of vote-codes sheets
- Step towards E2E verifiability
Motivations to build such a working model:

- No trust in the voting device
- Coercion resistance
- No trust in the generation/sending of vote-codes sheets
- Step towards E2E verifiability
Code-voting

(1342)

VC1

VC2

VC3

VC4

VC2 (Brad)

VC4 (Orw)

VC1 (Hem)

VC3 (Hess)

VeC1

VeC2

VeC3

VeC4

A different approach to code-voting
A different approach to code-voting
Motivations to build such a working model:

- No trust in the voting device
- Coercion resistance
- No trust in the generation/sending of vote-codes sheets
- Step towards E2E verifiability
Motivations to build such a working model:

- No trust in the voting device
- Coercion resistance
- No trust in the generation/sending of vote-codes sheets
- Step towards E2E verifiability
However, this construction has several pitfalls:

- Complexity of working with encrypted matrices and their related NIZKP
- The second product is between an encrypted matrix and an encrypted vector - homomorphic encryption not enough
- The matrix is encrypted with the VK, meaning that the key decrypting the matrix is the same as the one decrypting the verification code
- Specific properties are not satisfied (Dispute freeness or forced abstention)
However, this construction has several pitfalls:

- Complexity of working with encrypted matrices and their related NIZKP
- The second product is between an encrypted matrix and an encrypted vector - homomorphic encryption not enough
- The matrix is encrypted with the VK, meaning that the key decrypting the matrix is the same as the one decrypting the verification code
- Specific properties are not satisfied (Dispute freeness or forced abstention)
However, this construction has several pitfalls:

- Complexity of working with encrypted matrices and their related NIZKP
- The second product is between an encrypted matrix and an encrypted vector - homomorphic encryption not enough
- The matrix is encrypted with the VK, meaning that the key decrypting the matrix is the same as the one decrypting the verification code
- Specific properties are not satisfied (Dispute freeness or forced abstention)
However, this construction has several pitfalls:

- Complexity of working with encrypted matrices and their related NIZKP
- The second product is between an encrypted matrix and an encrypted vector - homomorphic encryption not enough
- The matrix is encrypted with the VK, meaning that the key decrypting the matrix is the same as the one decrypting the verification code
- Specific properties are not satisfied (Dispute freeness or forced abstention)
Project Objectives

- A trade-off has to be made to achieve usability, E2E verifiability and coercion-resistance
- Adversarial model has to be formally defined
- Research on whether we can reduce the trust assumptions for the filtering stage
- Study the feasibility of the solution involving the permutation matrices
- Present a construction offering both proposals
A different approach to code-voting
Achieving coercion-resistance and individual verifiability without the assumption of a trusted channel

Iñigo Querejeta-Azurmendi

Minsait, Indra
Universidad Carlos III de Madrid

October 23, 2017


