Berner Fachhochschule - Technik und Informatik - RISIS

UniVote

A remote e-voting system for university elections in Switzerland

Eric Dubuis

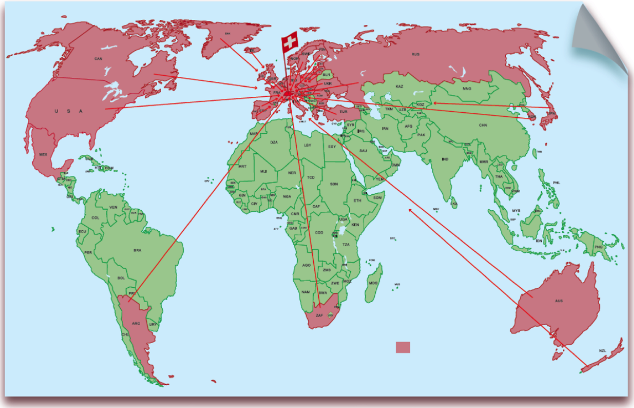
November 2, 2012

Current Situation in Switzerland (1)

Small country, three political levels, have to vote up to four times a year (elections: every 4 years):

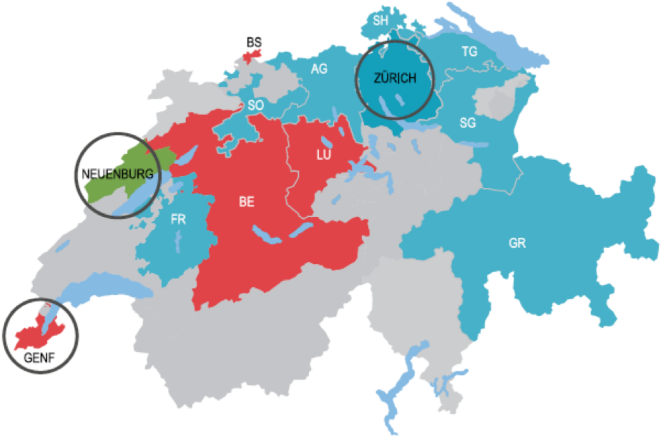
Current Situation in Switzerland (2)

Remote e-voting demanded mainly from expatriates:



Current Situation in Switzerland (3)

Three cantons (GE, NE, ZH) run e-voting systems, ten others use them jointly:



Why Do We Care?

In early 2008, we raised questions at persons in charge for Swiss e-voting systems such as:

- How is the secrecy of votes achieved?
- How is voter's privacy achieved?
- How is the integrity of votes achieved?
- How can ballot-box stuffing be avoided?
- How can the result be verified?
- ...

...and we didn't get any satisfactory answers (from a research point of view)!

Who Are We?

- Berner Fachhochschule is a university of applied sciences (approx. 6,000 students)
- We belong to the Engineering and Information Technology department
- The E-Voting group belongs to the Research Institute for Security in the Information Society (RISIS)
- The E-Voting group currently staffed with:
  - → 4 professors
    Rolf Haenni, Reto Koenig, Stephan Fischli, and myself
  - → 1 PhD candidate
  - → 1 research assistant
  - → 2 master students

Outline

Outline

Security Requirements for E-Voting Systems

- Correctness
  - → Democracy
    - eligible voters only (eligibility verifiability)
    - one voter, one vote that counts
  - → Integrity
    - after casting, votes cannot be altered, deleted, or substituted
  - → Accuracy
    - all valid votes are counted
    - invalid votes are not counted
- Privacy
  - → Secrecy: no one can tell how a voter voted
  - → Anonymity: no one can tell who voted
  - → Receipt-freeness: no one can prove whether or how she voted
  - → Fairness: no one can infer partial results before the election is closed

Security Requirements for E-Voting Systems

- Verifiability
  - → Individual verifiability
    - cast as intended
    - recorded as cast
    - counted as recorded
  - → Universal verifiability
    - anyone can verify the correctness of the election result

Outline

UniVote Facts

- Clients:
  - → University of Zurich
  - → University of Bern
  - → Berner Fachhochschule
- Due time (first version): March 2013
- WSDL component interface definitions
- Server components in Java, Java EE
- Voter client in Javascript

More Facts

They have:

- elections for deputies, president, etc.
- parties, lists, candidates
  - → candidates can be cumulated
  - → candidates from other lists can be added (vote-splitting)
- period of term: one year (Uni ZH), two years (Uni Bern, BFH)
- have yet-another-web application

Additional Requirements

They require:

- SWITCHaai/Shibboleth (www.switch.ch)
- "vote and go"

Our goals as researchers:

- demonstrated the features of a verifiable e-voting system
- and a few more...

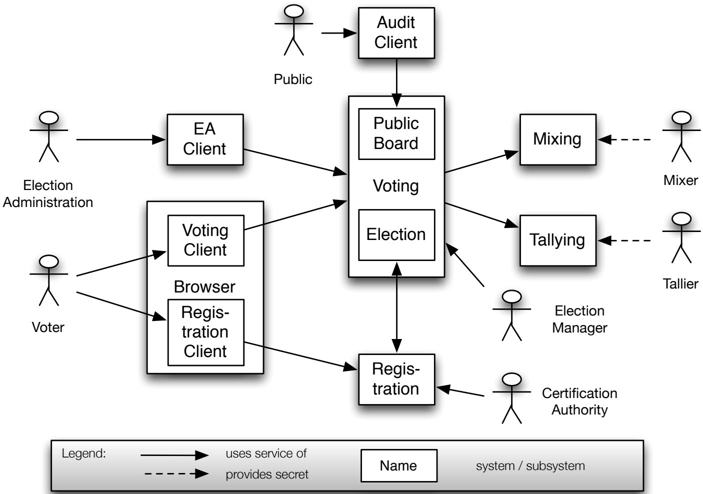
Non-Goals

From the set of requirements listed earlier, we exclude:

- that the solution is coercion resistant, and
- that the solution the secure platform problem

We do also not address the everlasting privacy problem.

System Overview



Outline

ElGamal Cryptosystem

Ingredients:

- Multiplicative cyclic group $(G_q, \cdot, 1)$ of order q.
- Typical choice:
  Subgroup of quadratic residues $G_q \subset \mathbb{Z}_p^*$ of prime order q, where $p = 2q + 1$ is a *safe prime*.
- Public parameters are thus p, q, and a generator g of $G_q = \langle g \rangle$

(x, y) is an ElGamal key pair, where $x \in_R \mathbb{Z}_q$ is private decryption key and $y = g^x \in G_q$ the corresponding public encryption key.

- Encryption of $m \in G_q$:
  $Enc_y(m, r) = (g^r, m \cdot y^r) \in G_q \times G_q$
- For a given $E = (a, b) = Enc_y(m, r)$, m can be recovered:
  $Dec_x(E) = a^{-x} \cdot b = m$

Homomorphic Property of ElGamal

The ElGamal encryption function is *homomorphic* with respect to multiplication:

- $Enc_y(m_1, r_1) \cdot Enc_y(m_2, r_2) = Enc_y(m_1 \cdot m_2, r_1 + r_2)$

Thus, a given encryption $E = Enc_y(m, r)$ can be *re-encrypted* by multiplying E with an encryption of the neutral element 1:

- $ReEnc_y(E, r') = E \cdot Enc_y(1, r') = Enc_y(m, r + r')$

This is an re-encryption of m with a fresh randomization $r + r'$.

Plaintext Encoding and Decoding

Plaintext needs to be selected from $\mathbb{Z}_q$ rather than G_q. With a safe prime p, we can use the following mapping $G : \mathbb{Z}_q \to G_q$ to encode any integer plaintext $m' \in \mathbb{Z}_q$ by a group element $m \in G_q$:

$$m = G(m') = \begin{cases} m' + 1, & \text{if } (m'+1)^q = 1, \\ p - (m'+1), & \text{otherwise.} \end{cases}$$

Given $m \in G_q$, we can reconstruct $m' \in \mathbb{Z}_q$ by applying the inverse function $G^{-1} : G_q \to \mathbb{Z}_q$ to m:

$$m' = G^{-1}(m) = \begin{cases} m - 1, & \text{if } m \leq q, \\ (p - m) - 1, & \text{otherwise.} \end{cases}$$

Schnorr Signatures (1)

Ingredients:

- Multiplicative cyclic group $(G_q, \cdot, 1)$ of order q.
- Typical choice:
  Schnorr group, a subgroup $G_q \subset \mathbb{Z}_p^*$ of prime order q, where $p = kq + 1$ is a large prime.
- Public parameters are thus p, q, and a generator g of $G_q = \langle g \rangle$
- Cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_q$

Schnorr Signatures (2)

An Schnorr signature key pair is a tuple (sk, vk), where $sk \in_R \mathbb{Z}_q$ is the randomly chosen private signature key and $vk = g^{sk} \in G_q$ the corresponding public verification key.

Let $m \in \{0,1\}^*$ denote an arbitrary message to sign, and $r \in_R \mathbb{Z}_q$ a randomly selected value, then the Schnorr signature for m is:

$$Sign_{sk}(m, r) = (a, r - a \cdot sk) \in \mathbb{Z}_q \times \mathbb{Z}_q, \text{ where } a = H(m||g^r)$$

Given a public verification key vk and a signature $S = (a, b) = Sign_{sk}(m, r)$ for message m, it can be verified by computing:

$$Verify_{vk}(m, S) = \begin{cases} accept, & \text{if } a = H(m||g^b \cdot vk^a), \\ reject, & \text{otherwise} \end{cases}$$

Zero-Knowledge Proofs of Knowledge

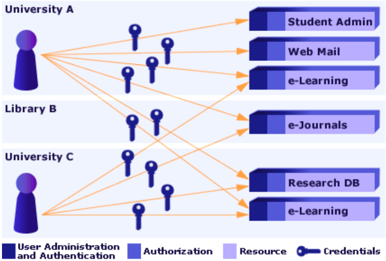
A *zero-knowledge proof* is a cryptographic protocol, where the *prover* P tries to convince the *verifier* V that a mathematical statement is true, but without revealing any information other than the truth of the statement.

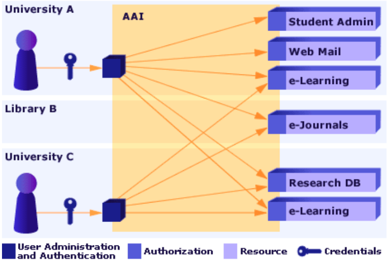
A *proof of knowledge* is a particular proof allowing P to demonstrate knowledge of a secret information involved in the mathematical statement. Notion for non-interactive variant:

$NIZKP\{(s_1, s_2, \ldots, s_n) : \text{relations among parameters and } s_i\}$

Outline

Voter Registration (1)





See also: www.switch.ch/aai

Voter Registration (2)

The public parameters p, $q = (p-1)/k$, and g for Schnorr signatures are known in advance and do not to change over time.

Person V_i performs the following steps:

1. Choose $sk_i \in_R \mathbb{Z}_q$ uniformly at random.
2. Compute $vk_i = g^{sk_i} \bmod p$.
3. Generate $\pi_{sk_i} = NIZKP\{(sk_i) : vk_i = g^{sk_i} \bmod p\}$ to prove knowledge of sk_i.
4. Send $(V_i, cred_i, vk_i, \pi_{sk_i})$ to CA.

vk_i is the public key for Schnorr signatures of voter V_i.

Voter Registration (3)

CA performs the following steps:

1. Check validity of $(V_i, cred_i)$.
2. Check correctness of π_{sk_i}.
3. Determine current timestamp t_i.
4. Compute $Z_i = Certify_{sk_{CA}}(V_i, vk_i, t_i) = (V_i, vk_i, t_i, CA, C_i)$.
5. Publish Z_i in public certificate directory (append-only).

Note that vk_i is the public (signature) key of voter V_i.

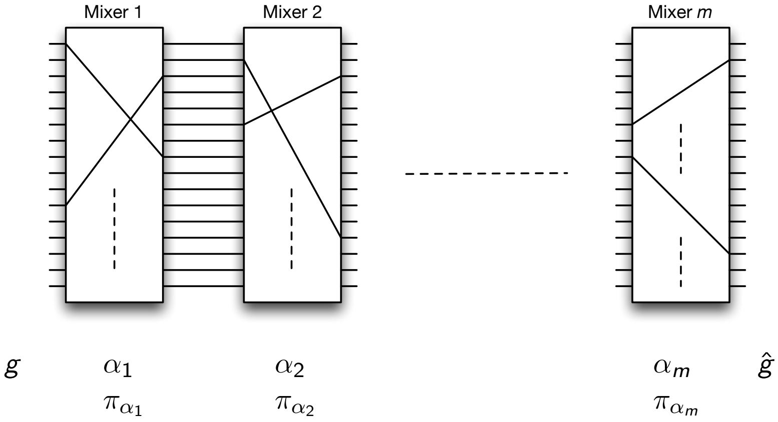
Registration Subsystem

The Registration subsystem publishes the public parameters p, $q = (p-1)/k$, and g for Schnorr signatures as well as the certificates of registered persons in an (append-only) manner:

Identifier V_i	Name, ...	Public key vk_i
...	...	...
...	...	...
314	Miller, ...	27983
722	Moore, ...	48094
...	...	...
...	...	...

Outline

Election Generator Construction (1)



Election Generator Construction (2)

Let $g_0 = g$ the publicly known generator of the Schnorr signature scheme. Each $M_k \in M$ performs the following steps:

1. Choose $\alpha_k \in_R \mathbb{Z}_q$ at random.
2. Compute blinded generator $g_k = g_{k-1}^{\alpha_k} \bmod p$.
3. Generate $\pi_{\alpha_k} = NIZKP\{(\alpha_k) : g_k = g_{k-1}^{\alpha_k} \bmod p\}$ to prove knowledge of α_k.
4. Generate signature $S_{g_k} = Sign_{sk_k}(id||g_k||\pi_{\alpha_k})$.
5. Publish $(M_k, id, g_k, \pi_{\alpha_k}, S_{g_k})$ on EB.

Election manager EB checks all proofs and publishes:

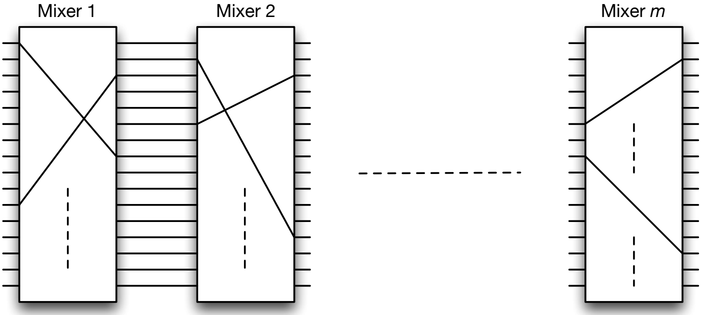
1. Let $\hat{g} = g_m$ be the *election generator*.
2. Publish $\hat{g}$ on EB.

Electoral Roll Preparation

- The Election Authority defines the set of eligible voters $V = \{V_1, \ldots, V_n\}$.
- For every voter V_i, select the most recent certificate $Z_i = (V_i, vk_i, t_i, CA, C_i)$ from the public certificate directory and verify it.

Recall that vk_i is the public key for Schnorr signatures of voter V_i.

Generating the Public Verification Keys (1)



g	α_1	α_2	α_m	$\hat{g}$
vk_i	ψ_1	ψ_2	ψ_m	$vk'_{\psi(i)}$
	π_{ψ_1}	π_{ψ_2}	π_{ψ_m}	
	π_{α_1}	π_{α_2}	π_{α_m}	

Generating the Public Verification Keys (2)

Let $VK_0 = \{vk_1, \ldots, vk_n\}$ be the (ordered) set of public keys in electoral roll $\mathcal{Z}_V$. Repeat the following steps for each mixer $M_k \in M$:

1. Shuffle the public keys VK_{k-1} into VK_k:
   1.1 Compute blinded key $vk'_i = vk_i^{\alpha_k}$ for every $vk_i \in VK_{k-1}$.
   1.2 Choose a permutation $\psi_k : [1, n] \rightarrow [1, n]$ at random.
   1.3 Let $VK_k = \{vk'_{\psi_k(i)} : 1 \leq i \leq n\} = Shuffle_{\psi_k}(VK_{k-1}, \alpha_k)$ be the new (ordered) set of public keys shuffled according to ψ_k.
2. Generate $\pi_{\psi_k} = NIZKP\{(\psi_k, \alpha_k) : g_k = g_{k-1}^{\alpha_k} \wedge VK_k = Shuffle_{\psi_k}(VK_{k-1}, \alpha_k)\}$ using Wikstroem's proof of a shuffle.
3. Generate signature $S_{VK_k} = Sign_{sk_k}(id || VK_k || \pi_{\psi_k})$.
4. Publish $(M_k, id, VK_k, \pi_{\psi_k}, S_{VK_k})$ on EB.

Encryption Key Generation

Election manager EM defines ElGamal parameters P, $Q = (P-1)/2$, and G.

Each Tallier $T_j \in T$ performs:

1. Choose $x_j \in_R \mathbb{Z}_Q$ uniformly at random.
2. Compute $y_j = G^{x_j} \bmod P$.
3. Generate $\pi_{x_j} = NIZKP\{(x_j) : y_j = G^{x_j} \bmod P\}$ to prove knowledge of x_j.
4. Publish signed value of y_j and proof π_{x_j} on EB.

Election manager EM computes $y = \prod_j y_j \bmod P$ and publishes signed value y on EB.

Value y will be used for encrypting the ballots.

Outline

Vote Creation and Casting

To cast a vote, voter $V_i \in V$ performs:

1. Retrieve election data from Election Board EB.
2. Validate signatures.
3. Determine $\mathcal{V}^* = Votes(C, R)$ election options.
4. Choose vote $v_i \in \mathcal{V}^*$.
5. Represent v_i as an integer $m'_i = Encode_{C,R}(v_i) \in \mathbb{Z}_Q$.
6. Compute $m_i = G(m'_i) \in G_Q$.
7. Choose $r_i \in_R \mathbb{Z}_Q$ uniformly at random.
8. Compute $E_i = Enc_y(m_i, r_i) = (a_i, b_i)$.
9. Compute anonymous verification key $vk'_j = \hat{g}^{sk_i}$, where $j = \psi(i)$.
10. Generate π_{r_i} to prove knowledge of (m_i, r_i).
11. Generate signature $S_i = Sign_{sk_i}(id||E_i||\pi_{r_i})$ using $\hat{g}$.
12. Send ballot $B_i = (vk'_j, id, E_i, \pi_{r_i}, S_i)$ to EB.

Vote Recording and Publishing

Upon receipt of B_i, Election manager EB checks:

1. Check that vk'_j is V_i's most recent key.
2. Check that $Verify_{vk'_j}(id||E_i||\pi_{r_i}, S_i) = accept$ using $\hat{g}$.
3. Check that V_i has not previously submitted another ballot:[1]
   3.1 Check that no ballot on EB contains vk'_j.
   3.2 If $vk'_j \in \bar{VK}'$, check that no ballot on EB contains a former key $\hat{vk}'_i \in \hat{VK}'$ of V_i.
4. Optional: Check correctness of π_{r_i}.

B_i is published, if all tests succeed.

[1] Since re-voting is not supported, only the first ballot counts.

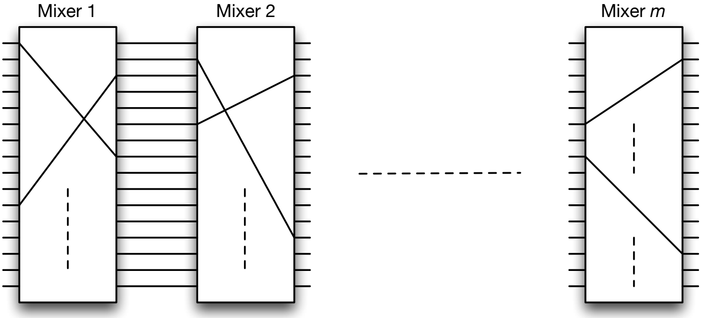
Closing the Electronic Urn

Upon closing the electronic urn, the Election Manager EM performs:

1. For each $B_i = (vk'_j, id, E_i, \pi_{r_i}, S_i)$, do the following:
   1.1 Check that $vk'_j \in VK'$.
   1.2 Check that $Verify_{vk'_j}(id||E_i||\pi_{r_i}, S_i) = accept$ using $\hat{g}$.
   1.3 Check correctness of π_{r_i}.
2. Let $\mathcal{B}$ be the set of ballot B_i, for which all above checks succeed.
3. Generate signature $S_{\mathcal{B}} = Sign_{sk_{EM}}(id||\mathcal{B})$.
4. Publish $(EM, id, \mathcal{B}, S_{\mathcal{E}})$ on EB.

Outline

Mixing the Encryptions (1)



$\mathcal{E}_0$ $\mathcal{E}_1$ $\mathcal{E}_2$ $\mathcal{E}_m$

$\bar{r}_1$ $\bar{r}_2$ $\bar{r}_m$

$\mathcal{T}_1$ $\mathcal{T}_2$ $\mathcal{T}_m$

π_1 π_2 π_m

Mixing the Encryptions (2)

Let $\mathcal{E}_0 = \{E_1, \ldots, E_N\}$, $N \leq n$, be the (ordered) set of encrypted votes in $\mathcal{B}$. For each Mixer $M_k \in M$:

1. Shuffle the encrypted votes $\mathcal{E}_{k-1}$ into $\mathcal{E}_k$:
   1.1 Choose $\bar{r}_k = (r_{1k}, \ldots, r_{Nk}) \in_R \mathbb{Z}_q^N$ uniformly at random and compute $E'_i = ReEnc_y(E_i, r_{ik})$ for every $E_i \in \mathcal{E}_{k-1}$.
   1.2 Choose permutation $\tau_k : [1, N] \to [1, N]$ uniformly at random.
   1.3 Let $\mathcal{E}_k = \{E'_{\tau_k(i)} : 1 \leq i \leq N\} = Shuffle_{\tau_k}(\mathcal{E}_{k-1}, \bar{r}_k)$ be the new (ordered) set of encrypted votes shuffled according to τ_k.
2. Generate $\pi_k = NIZKP\{(\tau_k, \bar{r}_k) : \mathcal{E}_k = Shuffle_{\tau_k}(\mathcal{E}_{k-1}, \bar{r}_k)\}$ using Wikstroem's proof of a shuffle.
3. Generate signature $S_{\mathcal{E}_k} = Sign_{sk_k}(id||\mathcal{E}_k||\pi_k)$.
4. Publish $(M_k, id, \mathcal{E}_k, \pi_k, S_{\mathcal{E}_k})$ on EB.

Mixing the Encryptions (3)

Finally, the Election Manager EM performs:

1. For each $M_k \in M$:
   1.1 Check that $Verify_{vk_k}(id||\mathcal{E}_k||\pi_{\tau_k}, S_{\mathcal{E}_k}) = accept$
   1.2 Check correctness of π_{τ_k}.
2. Let $\mathcal{E}' = \mathcal{E}_m = \{E'_{\tau(i)} : 1 \leq i \leq N\}$ for $\tau = \tau \circ \cdots \circ \tau_1$.
3. Generate signature $S_{\mathcal{E}'} = Sign_{sk_{EA}}(id||\mathcal{E}')$.
4. Publish $(EM, id, \mathcal{E}', S_{\mathcal{E}'})$ on EB.

$\mathcal{E}'$ denote the re-encrypted and mixed votes.

Decrypting the Votes

Each $T_j \in T$ knows its private key share x_j and performs the following steps:

1. Check that $Verify_{vk_{EM}}(id||\mathcal{E}', S_{\mathcal{E}'}) = accept$.
2. Let $\bar{a} = (a_1, \ldots, a_N)$ for $(a_i, b_i) \in \mathcal{E}'$.
3. Compute $\bar{a}_j = (a_{1j}, \ldots, a_{Nj})$, where $a_{ij} = a_i^{-x_j} \bmod P$.
4. Generate π'_{x_j} to prove knowledge of x_j and the correct decryption of a_{ij} with x_j.
5. Generate signature $S_{\bar{a}_j} = Sign_{sk_j}(id||\bar{a}_j||\pi'_{x_j})$.
6. Publish $(T_j, id, \bar{a}_j, \pi'_{x_j}, S_{\bar{a}_j})$ on EB.

Decoding the Votes

Votes are decrypted now, but still encoded. The Election Manager EM checks signatures, proofs, and decodes the encoded votes:

- For all $1 \leq i \leq N$, do the following:
  1. Compute $m_i = b_i \cdot \prod_j a_{ij} \bmod P$.
  2. Compute $m'_i = G^{-1}(m_i)$.
  3. Compute $v_i = Decode_{C,R}(m'_i)$.
- Let $\mathcal{V} = \{v_1, \ldots, v_N\} \cap \mathcal{V}^*$ be the list of valid plaintext votes.
  1. Generate signature $S_{\mathcal{V}} = Sign_{sk_{EM}}(id||\mathcal{V})$.
  2. Publish $(EM, id, \mathcal{V}, S_{\mathcal{V}})$ on EB.

Plaintext votes can be counted now.

Outline

Conclusion and Future Work

Current status:

- still in the implementation phase...
- a little bit behind schedule
- ...spent a lot of time in developing a crypto library

Things to do later:

- threshold crypto system for talliers
- "bullet-proof" append-only public bulletin board
- distributed append-only public bulletin board

Thank You

?

Contact

Eric Dubuis <eric.dubuis@bfh.ch>
E-Voting Group: e-voting.ti.bfh.ch
RISIS: ti.bfh.ch/risis
BFH-TI: ti.bfh.ch