Running mixnet-based elections with Helios

Philippe Bulens – Damien Giry – *Olivier Pereira*

EVT/WOTE'11

Helios

- Open-audit elections from your browser
  - ubiquitous but computationally limited voting client
- Low-coercion elections
  - hard to separate voter from coercer in a remote setting
  - also true for mixnet-based tallying
- More and more experiences: > 40000 votes tallied
  among which ≈ 8400 through mixnets
  (the others using homomorphic tallying)

Homomorphic vs. mixnet tallying

Homomorphic tallying

- Public aggregation of ballots into election outcome ☺
- Trustees decrypt outcome only
  - little info. revealed ☺
  - little computation needed ☺
- ZK proofs of ballot validity
  - lot of computation ☹
  - need changes depending on election rules ☹
  - validity can be checked at submission time ☺

Burden on voters and programmers

Mixnet based tallying

- Trustees anonymise ballots ☹
- Trustees decrypt all shuffled ballots
  - more info. revealed ☹
  - computation grows with number of voters ☹
- Validity checked after decryption
  - no validity proof needed ☺
  - universal ballot format ☺
  - invalid ballots hard to trace ☹

Burden on trustees and election organizers

UCL Student elections

AGL (the UCL student association), Sep. 2009:

“Could we have verifiable elections on the Internet?”

- “Well, how do your elections work?”

- “We typically have $\approx$ 250 candidates, organized in lists (parties), and voters can select as many candidates they like as long as they are from the same list”

That killed the homomorphic tallying approach for current JavaScript crypto implementation performance:

- ZK proofs of ballot validity
  - lot of computation ☹

Running mixnet-based elections with Helios

What do we need to change?

1. Election setup?
   Not really:
   - Make mixing trustees independent (reencryption mixnet)
   - Keep same key management
2. Ballot preparation?
   Yes:
   - one ciphertext per question, no validity proof
   - ciphertexts need to be proven independent
3. Audit and tally procedure?
   Yes:
   - Mixing is a new task
   - Decryption becomes a computationally intensive task
   - Decryption must be followed by validity verification and counting

Ballot preparation

1. Voters need to encrypt their choices using a randomizable scheme
2. Voters need to show that their ciphertexts are independent of others

How to reconcile these goals?

Wikström ['06] proposed *submission secure augmented cryptosystems*:

- Take a *basic* cryptosystem, randomizable in our case
- *Augment* it into a non-malleable (CCA2) cryptosystem
- Have a *strip* procedure that:
  - enables public verification of the CCA2 ciphertext correctness
  - allows extracting the embedded basic ciphertext

Resulting procedure:

1. Voters encrypt their choices with augmented cryptosystem
2. Server rejects duplicate ciphertexts
3. Strip augmented ciphertexts into randomizable ciphertexts
4. Mix those randomizable ciphertexts

Selecting an SSA cryptosystem

Choice criteria:

- Efficient solution
- Do not degrade Helios computational model (DDH, random oracle)

Candidates:

- ElGamal + Schnorr PoK of randomness

$$\boxed{g^y, v \cdot h^y} \qquad \boxed{g^r, e, r + e \cdot y}$$

  - Efficient: 3 modexp/ciphertext, vote independent
  - But not known to be CCA secure under DDH in RO model

- Double ElGamal (Naor-Yung) + Proof of identical ciphertexts

$$\boxed{g^y, v \cdot h^y} \qquad \boxed{g^z, v \cdot h'^z, \quad g^r, g^s, h^r \cdot h'^s, e, r - e \cdot y, s + e \cdot z}$$

  - Less efficient: 8 modexp/ciphertext, vote independent
  - Known to be CCA secure under DDH in RO model

Selecting an SSA cryptosystem

Choice criteria:

- Efficient solution
- Do not degrade Helios computational model (DDH, random oracle)

More Candidates:

- Cramer-Shoup encryption (advocated by [Wik06])

$$\boxed{g^y, v \cdot h^y} \quad \boxed{g'^y, c^y \cdot d^{y \cdot H(g^y, v \cdot h^y, g'^y)}}$$

  - Fairly efficient: 5 modexp/ciphertext, 1 is vote dependent
  - CCA secure under DDH in the standard model!
  - needs to reveal secret values used to generate c and d to check ciphertext validity

Used to tally 4488 votes in March 2010 (out of $\approx$ 26000 potential):

- worked fine, but ... annoying in practice
- ballot independence can only be checked after election closing
- handling c and d adds burden on the trustees

Selecting an SSA cryptosystem

Choice criteria:

- Efficient solution
- Do not degrade Helios computational model (DDH, random oracle)

More Candidates:

- Variant of TDH2 scheme [SG97] with homomorphic basic scheme

$$\boxed{g^y, v \cdot h^y} \quad \boxed{g'^y, g^r, g'^r, e, r + e \cdot y}$$

  - Fairly efficient: 5 modexp/ciphertext, vote independent
  - CCA secure under DDH in RO model

Used to tally 3951 votes in April 2011 (out of $\approx$ 26000 potential):

- Much more comfortable in practice

Mixing ballots

Selection criteria:

- Efficient solution
  Many available: Furukawa et al., Groth, Neff, Wikström, ...
- Simple concepts, use expected to not be restricted by patents:
  Terelius, Wikström '09,'10

Usage:

- 3 shuffling trustees interacting through voting server:
  - collecting ballots
  - uploading shuffled ballots and proofs
  - verifying other people's proofs
- Using single-file python script based on standard libraries
  - performances good enough: $\approx$ 25 ballots shuffled/sec.
- If you need something more complete and efficient:
  Verificatum!

Lessons learned

1. Trustees manipulate sensitive data privately
   $\Rightarrow$ Keep their job as simple as possible

2. Organizers and voters expect results quicky
   Tally orchestration was more time-consuming than computation
   $\Rightarrow$ Solve as much organisational problems as possible before tally

3. Mixnet-based tallying works fine and is quite general but. . .
   if you can use homomorphic tallying, go for it!