Pretty Good Voting (PGV)

Christian Bell, Jason Duell, Amir Kamil

Computer Security

CS 261

Fall 2004

PGV Introduction

- According to the SERVE report, *"there really is no good way to build [..] a voting system without a radical change in the overall architecture of the Internet and the PC, or some unforeseen security breakthrough"*

- PGV is an effort to provide a practical Internet voting solution

- What's the best we can do with current Internet technologies assuming we are not targeting the holy grail of elections, presidential elections?

- How and to whom can we provide 'Pretty Good Voting' ?
  - Elections possible for non-profit organizations, corporate shareholders
  - Potential for higher voter turnouts
  - Potential for higher voter convenience
  - Potential for higher confidence in results

Problem space

Application

Federal online elections

PGV

E-commerce
(HTTPS)

Level of difficulty

“Impossible”

? ? ?

Easy and well
understood

Election Requirements

- “Must-haves”
  - Fair count: registered voters only, vote only once, counted accurately
- “Nice to Have”
  - “Strong” anonymity
    - “weak”, ecommerce-style anonymity may be OK
- Not needed?
  - Preventing coercion, selling of votes
  - Receipt-free (receipts are good!)
  - Denial of Service (temporary DoS is OK)

Focus on Feasibility/Acceptance

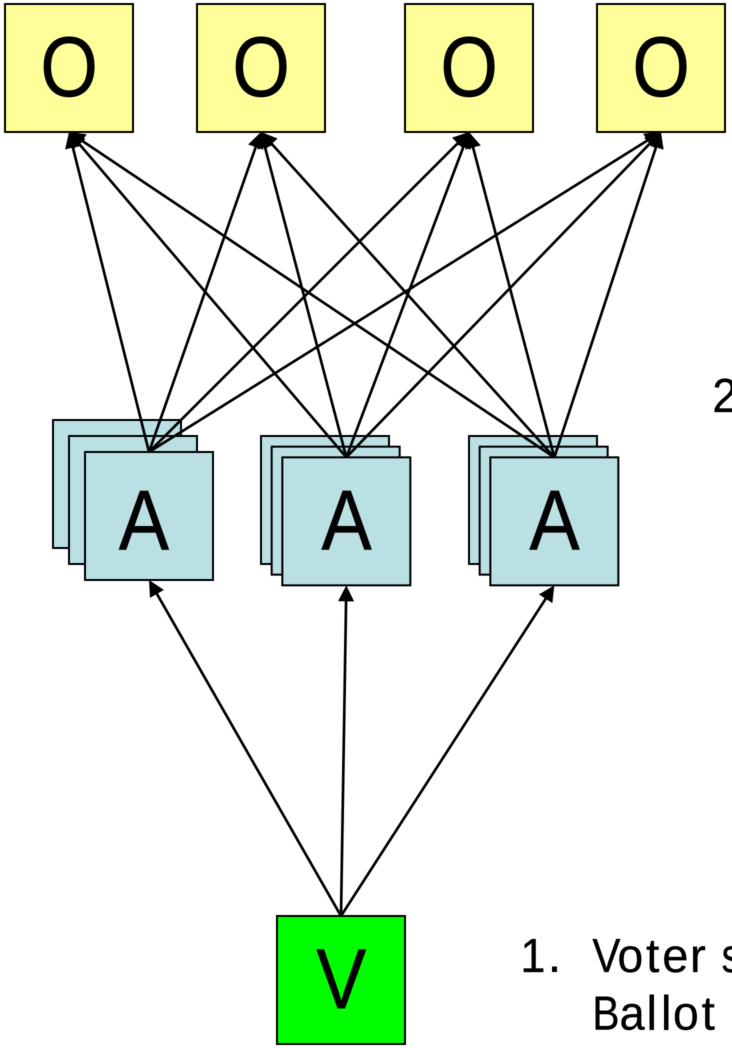
PGV Environment

- Registered voters only
- Collusion resistance:
  - Decentralized tabulation
  - Prevent ballot stuffing
- Robust: don’t lose votes
- Spyware detection
- E-commerce style *availability* (web server) and *security* (SSL and DNS): *No better, no worse*
- Open policy: open security and voting protocols

Ease of Use

- Standard Web browser
  - Perhaps with applet, plugin
- No user key management
- Voters can see their ballots (in plain text) in the results

PGV "simple" solution

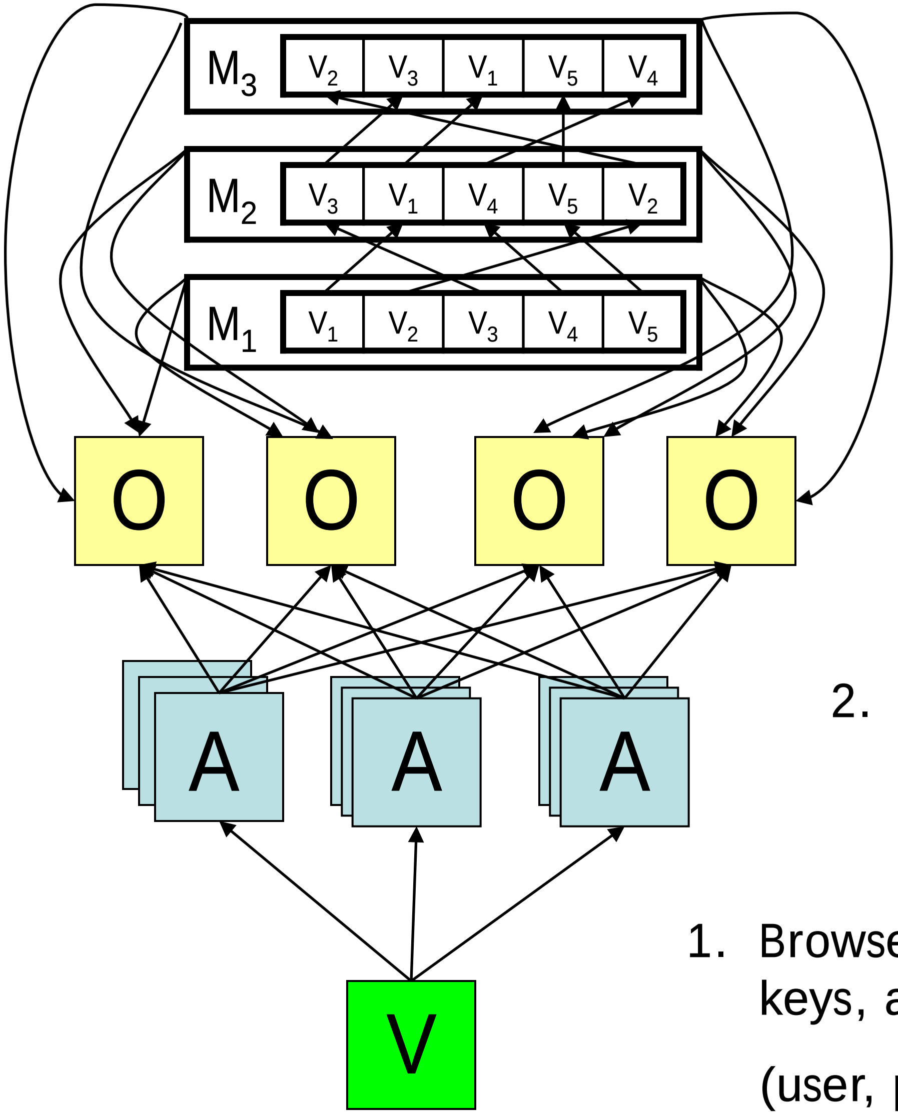


3. Observers check that authentication servers produce same results. Publish votes. Voters can find/check their ballot via their unique_id.

2. Authentication servers validate user, then send ballot to Observers with voter obscured as MD5(user, password).

1. Voter sends ballot to all authentication servers: Ballot = $\{\text{User, Pass, Vote, unique\_id}\}_{\text{Kas}}$

PGV Mix-net solution



M_3
V_2 V_3 V_1 V_5 V_4
M_2
V_3 V_1 V_4 V_5 V_2
M_1
V_1 V_2 V_3 V_4 V_5
O
O
O
O
A
A
A
V

4. Mix-net shuffles voters/votes. Each step stored with observers for verfiability, including final results. Voters can find/check their ballot via their unique_id.

3. Observers check that authentication servers produce same ciphertexts; pass into mix-net.

2. Authentication servers validate user, sign {vote} and send to observers.

1. Browser encrypts ballot with Mix-net public keys, and sends to authentication servers:

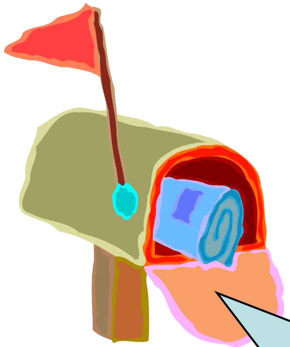
   (user, password, $\{\{\{\text{vote, unique\_id}\}_{M3}\}_{M2}\}_{M1}$)

Security guarantees

- Authentication servers
  - Must all produce same result, or flag raised, so all must collude to tamper with votes
  - Sign results, so fraud traceable
- Observers
  - Not trusted with any secrets. All inputs signed by source, so can’t tamper.
- Mix-Net
  - All servers would need to collude to compromise voter’s anonymity.

“Spyware” detection

- Use out-of-band channel to distribute per-voter permutations.
- Voters cast ballot for symbol corresponding to candidate.
- Spyware can’t predict symbol for a given candidate, so can’t swing election (at best can randomly misrepresent voter)

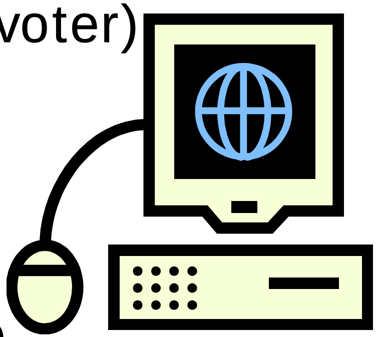


1. Mail ballot

🍊 = Kerry

🍋 = Bush

🍉 = Nader



2. On-screen vote

1. 🍋
2. 🍉
3. 🍊