LYU1803:
Opensource E-voting System for 8 million mobile devices

ESTR4999 Graduation Thesis Presentation

Maxwell Chan presents

supervised by Prof. Michael Lyu
Review of Term 1
Motivation

- Disadvantage of paper-based voting
- Blockchain
- End-to-end verifiable voting
Design

- Helios

- Permissioned blockchain
Implementation

- Election encryption & decryption
- “Backbone” of blockchain
- Draft user interface
Objective
Overall schedule

Objective

Term 1
- Research
- Design
- Implementation

Term 2
- Testing
Objective of Term 2

- Zero-knowledge proof

- Communication full verification

- User interface design

- Load test
Protocol design
Protocol design

Zero-knowledge proof

- Sigma protocol
- Non-interactive mode
Trustee knowledge on private key

- Need all private key for decryption

- Malicious example:

  Trustee public key: $y_i$  private key: $x_i$

  Submit public key as: $y_i/(y_1y_2y_3...y_n)$

  Election public key = $(y_1y_2y_3...y_n) \times y_i/(y_1y_2y_3...y_n) = y_i$
Trustee honest decryption

- Must use the private key
- Malicious example: $x_i g^1$ or $x_i g^{-1}$
- Ballot aggregation
Voter honest encryption

- Encrypt only 0 or 1
- Limit number of selection
- Use “simulated proof” for other values
  - Reverse the Sigma protocol
  - Verify sum of “Challenge”
Protocol design

Authentication method

- Signature bound with ballot
- Key generation problem
  - Server generate, send via email
  - Voter self-enrollment
  - Election administrator upload directly
Roles and permission

Different type of administrator:

- Server administrator
- Election administrator
- Trustee
Protocol design >> Blockchain

Block design

1. Election details
2. Ballots
3. Election tally
Block generation - Node selection

- Use Server ID instead of address
- Unique ID for each server key pair

<table>
<thead>
<tr>
<th>IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>3000</td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>3001</td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>5000</td>
</tr>
<tr>
<td>127.0.0.2</td>
<td>3000</td>
</tr>
<tr>
<td>127.0.0.3</td>
<td>3000</td>
</tr>
</tbody>
</table>

6789 mod 5 = 4
4 + 1 = 5

<table>
<thead>
<tr>
<th>Server ID</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>g8enw62QuO</td>
</tr>
<tr>
<td>2</td>
<td>12w9Kp/hBb</td>
</tr>
<tr>
<td>3</td>
<td>eUO+IHv9O2</td>
</tr>
<tr>
<td>4</td>
<td>ExmMM/5d1B</td>
</tr>
<tr>
<td>5</td>
<td>Zt5bEEXYKc</td>
</tr>
<tr>
<td>6</td>
<td>50nyQ8iBl4</td>
</tr>
</tbody>
</table>
Implementation
Implementation

System architecture

- Modularized design
Demo

1. Vote in a prepared election
2. Tally the election
3. Decrypt the election
4. Show result
Testing
Overview

- Aim: Bottleneck of scaling up
- Load test (2 round)
  - Block length test
  - Arrival rate test
  - Ballot aggregation test
- Reliability test
Environment

- CSE machine x3
  - 4 CPU @ 2.8GHz
  - 8GB RAM

- Google Cloud Virtual Machine
  - 8 CPU @ 2.5GHz
  - 56GB RAM
Testing >> Load test >> First round

Block length test - CSE machine

- Case of 100,000 ballots
Block length test - CSE machine

- Case of 1,000,000 ballots
  - Database out of memory
  - Some ballots in multiple blocks
Block length test - Google VM

- Case of 1,000,000 ballots
- Processing time increase
Arrival rate test - 1 node - CSE machine

- 11 Ballots per second

- 12 Ballots per second
Arrival rate test - 1 node - Google VM

- 12 Ballots per second
Testing >> Load test >> First round

Ballot aggregation test - CSE machine

- 3 nodes on same machine
- 100 Ballots: 0.07 second
- 1000 Ballots: 0.6 second
- 10000 Ballots: 72 seconds
Testing >> Load test >> First round

Summary & Improvement

- Cannot scale up
- Single thread → Low CPU utilization
  - Fork child processes for ballot processing
Summary & Improvement

- High database memory usage → Long response time
  - More index
    - electionID ↑, blockType ↑, blockSeq ↓, data.voters.id ↑
    - electionID ↑, blockSeq ↓
Block length test - Google VM

- Case of 1,000,000 ballots
  - Crash at ~950,000 ballots

- Extended test
Arrival rate test - 1 node - Google VM

- **48 Ballots per second**

- **56 Ballots per second**
Arrival rate test - 1 node - Google VM

- 48 Ballots per second

- 56 Ballots per second
Testing >> Load test >> Second round

Arrival rate test - CSE machine

- 1 node
  - 30-31 Ballots per second

- 2 nodes
  - 28-29 Ballots per second
Ballot aggregation test

- With 1 or 2 node(s) on CSE machine
Summary & Improvement

- Great improvement
- Able to scale up
- Memory leakage problem
  - Fixed (on block generation)
Reliability test

- 3 nodes on the same CSE machine
- Able to adapt the situation
- Need time to sync
Conclusion
Overview of Term 2

- Zero-knowledge proof
- Revised design
  - Authentication
  - Blockchain
- Implementation
  - Full verification between nodes
  - User interface & authentication
- Testing & improvement
- Opensource
Possible application

Legislative Council Election

- 5 geographical constituencies (GC)
- 1 million voters per GC
- 100,000 votes per hour
- 28 ballots per second
Future work

- Improvement on scalability
- Improvement on reliability
- Full implementation of the proposed design
- Enforce more security measure
- Use newer communication protocols
- Possibility of enabling “Voting-as-a-service”
Conclusion >> Future work

**Improvement on scalability**

- More Child processes
  - Blockchain, Election, Handshake

- Partially broadcasting ballots
  - nodes with same database
Improvement on reliability

- Ballot re-broadcasting
  - Voter experience

- Smarter blockchain synchronization
  - Sequence of ‘invalid blocks’

- Clock synchronization
  - For block generation
Full implementation of the proposed design

- Kiosk voting

- Authentication method
Enforce more security measure

- Removed for the ease of testing
- Replay attack → Nonce
- Secure connection → HTTPS
Use newer communication protocols

- **Current**: All via HTTP
- **Improvement**: Some via TCP
- **Future**: Use QUIC
Possibility of enabling “Voting-as-a-service”

- Pay for computation power used
- Earn by hosting as a node