E-voting on DELA

Auguste Baum, Ambroise Borbely and Emilien Duc

DEDIS Lab - Prof. Bryan Ford
Supervisor: Noémien Kocher
Backend
Plan

❖ Our goals

❖ Main cha(lle)nges

❖ Evaluation

❖ Future work
Plan

❖ Our goals
  ➢ Project requirements
  ➢ Background

❖ Main changes

❖ Evaluation

❖ Future work
Project requirements

Security:
- Transparency/Auditability
- Resilience
- Vote secrecy
- Data integrity
- Availability

Roles:
- Admin
- Voters

Execution:

1. Create
2. Cast
3. Close
4. Shuffle
5. Decrypt

* Cancel

(time)
Plan

❖ Background
  ➢ Project requirements
  ➢ Background
    ■ Dela & smart contract
    ■ DKG
    ■ Neff Shuffle

❖ Main changes

❖ Evaluation

❖ Future work
Background: Dela & Smart contract

```plaintext
proc CreateElection:
  if CreateElectionTx:
    make(Election)
  exit
```

reads from Dela
Background: DKG
Background: Neff Shuffle
Background: All together

Tools:
- DKG
- Neff Shuffle

Smart contract methods

Init
Create
Cast
Close
Shuffle
Decrypt

* Cancel

time
Plan

❖ Our goals

❖ Main changes
  ➢ DKG
    ■ One DKG instance per election
    ■ Persistence of DKG credentials
  ➢ Neff Shuffle
  ➢ Election format

❖ Evaluation

❖ Future work
DKG: One DKG instance per election (1/3)
DKG: One DKG instance per election (2/3)
DKG: One DKG instance per election (3/3)
DKG: Persistence of DKG credentials

type DKGService struct {
    electionID   ID
    pubKey  PubKey
    secretPartKey  PrivKey
    rpc  RPC
    factory  serde.Factory
}
Plan

❖ Our goals

❖ Main changes
  ➢ DKG
  ➢ Neff Shuffle
    ■ Security issue
  ➢ Election format

❖ Evaluation

❖ Future work
Neff shuffle: Security issue

We want enough nodes to make a shuffle:

Algorithm:
1. All nodes shuffle the ballots and submit the result
2. One shuffle is accepted.
3. Start in 1 again but with the new shuffled ballots as input and without the node who made the accepted shuffle. Until enough shuffles are accepted.
Solution

- The nodes have to **sign** their shuffle

- **Refuse** shuffle if the node already has achieved one
Plan

❖ Our goals

❖ Main changes
  ➢ DKG
  ➢ Neff Shuffle
  ➢ Election format
    ■ Ballot size
    ■ Sequence shuffle
    ■ Election configuration

❖ Evaluation

❖ Future work
Election format: Ballot Size

- Kyber can only encrypt plaintexts < 29 bytes
  → split plaintext into **chunks**

![Diagram of ballot and encryption process]

**Encrypted Ballot**

- \[\text{byte(ElGamal pair)}\]
- \[\text{byte(ElGamal pair)}\]
- \[\text{byte(ElGamal pair)}\]
Election format: Ballot Size

- Type refactoring
- Adaptation of the protocols

Encrypted Ballot

- []byte(ElGamal pair)
- []byte(ElGamal pair)
- []byte(ElGamal pair)
Election format: Sequence shuffle

- Shuffle of ElGamal sequences
- Recent feature of kyber*

Layout of the ElGamal pairs in memory. \((X_{ij}, Y_{ij})\) is the \(j^{th}\) pair of the \(i^{th}\) ballot.

*alpha release, awaiting crypto review
Election format: Proving a shuffle of sequence

- The prover needs a **random** vector from the verifier
  → Problem of verifiable randomness

- The prover uses a **semi-random generator** to get the vector on its own
Election format: Election configuration (1/2)

- Bigger Ballots → More complex polls!

- 3 Types of Questions:
  - Ranked
  - Select
  - Open text

- A Subject groups multiple questions and sub-Subjects such that the Layout is fixed

```go
type Configuration struct {
    MainTitle string
    Scaffold  []Subject
}
```
**Election format: Election configuration (2/2)**

**Rank** your favorite foods:
1. Chocolate
2. Caramel
2. Raspberry
2. Orange
3. Licorice

**Choose** one hot drink:
- Cappuccino
- Latte
- Hot chocolate
- Espresso
- Flat White

**Write** down your name:

- First name
- Last name

```go
type Rank struct {
    ID ID
    Title   string
    MaxN    uint
    MinN    uint
    Choices []string
}
```
Plan

❖ Our goals

❖ Main changes

❖ Evaluation
  ➢ Correctness
  ➢ Performance

❖ Future work
Evaluation: Correctness

<table>
<thead>
<tr>
<th></th>
<th>cothory</th>
<th>last semester</th>
<th>now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart contract</td>
<td>37</td>
<td>87</td>
<td>76</td>
</tr>
<tr>
<td>DKG</td>
<td>65</td>
<td>90</td>
<td>88</td>
</tr>
<tr>
<td>Neff Shuffle</td>
<td>65</td>
<td>93</td>
<td>88</td>
</tr>
</tbody>
</table>

Test coverage evolution (%)

+ New integration tests added very recently
Evaluation: Performance (1/3)

- Focus on **shuffling** and **decryption**
- Parameters:
  - Number of nodes
  - Number of ballots in election
  - Size of ballots
Evaluation: Performance (2/3)

- Shuffle time is **linear** in number of nodes
- well...
- Number of chunks is fixed at 3 per ballot
Evaluation: Performance (3/3)

- Linear in log-log scale, hence decryption time is a **power law** of number of ballots

- Number of nodes fixed at 7

- Pairs = Chunks
Plan

❖ Our goals

❖ Main changes

❖ Evaluation

❖ Future work
  ➢ Stability
  ➢ Decryption
  ➢ Linking the backend and frontend
Future work: Stability

- More tests

- In more exotic situations

- With more nodes/ballots
Future work: Decryption

- Transparency
- Speed
Future work: Linking the backend and frontend

- Authentication of users
- Election formats
Frontend
Plan

❖ Tequila authentication

❖ Dela node request signatures

❖ Administration panel
How it was at the beginning

- One React process
- Anyone can login (and have a random ID) and create/manage elections and vote
Tequila authentication

- Need to add a new trusted backend (as React is only for frontend)
- Modification on the webpages
  - Show the user’s name on each page
  - Actually “log in” the users on React and Express processes
Authentication process

Client

GET request
POST request
data response
redirect response
webpage response

Express

/api/getTkKey
redirect_url
/key
redirect_url
tequila web form
gaspar + password
express/api/control_key?key=XXXX
/api/control_key?key=XXX
homepage + session cookie

Tequila

/cgi-bin/tequila/createresquest
gaspar + password
express/api/control_key?key=XXXX
/sciper, name, firstname, units
DELA message signature : reason

- Until now, everybody could send actions to the blockchain
- With Tequila implemented, actions must be now trusted / signed

→ Let all the traffic that goes to the DELA nodes pass through the Express
→ The Express adds current user data and signs the data
DELA message signature: architecture change

- The ExpressJS receives all requests that need to go to the DELA nodes
DELA message signature: signing process

1. Input payload
2. Base 64 encode
3. SHA256 hash
4. Schnorr Signature
5. Output payload
Administration Panel

- Since users are authenticated, we can set roles to users to allow / disallow certain actions:
  - Voter: can vote
  - Operator: can create / manage / close elections and can do the same as a voter
  - Admin: can add operators / admins and do the same as an operator
Administration Panel : User Interface

- Added a new view that allow admins to add a role to a user
- Changes the navigation bar to only display the correct tabs

View as an admin user
Administration Panel: Database

- Added a database with only one table to store roles

```
1   SELECT * FROM dedis.user_rights;
```

<table>
<thead>
<tr>
<th>id</th>
<th>sciper</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>231123</td>
<td>operator</td>
</tr>
<tr>
<td>6</td>
<td>111112</td>
<td>operator</td>
</tr>
<tr>
<td>7</td>
<td>123567</td>
<td>admin</td>
</tr>
<tr>
<td>12</td>
<td>295747</td>
<td>admin</td>
</tr>
</tbody>
</table>
Administration Panel : Backend access

- Middleware on the Express server that allow / reject a request depending on:
  - The user’s role
  - The current URL to access
Production-ready configuration

- Set up of a server with the following configurations
  - NGINX as a reverse proxy that holds the SSL certificate
  - Custom services files to run the different processes
  - crond configuration to restart the apps often
Demo
Conclusion

- Focused on security, made advances in usability

- Addressed many issues… and found new ones

- The project should be usable during the next semester!
The project’s journey

Where it was
Proof of concept

Where it is now
Refactoring, Enhancement, Testing

Where it’s going
Production ready