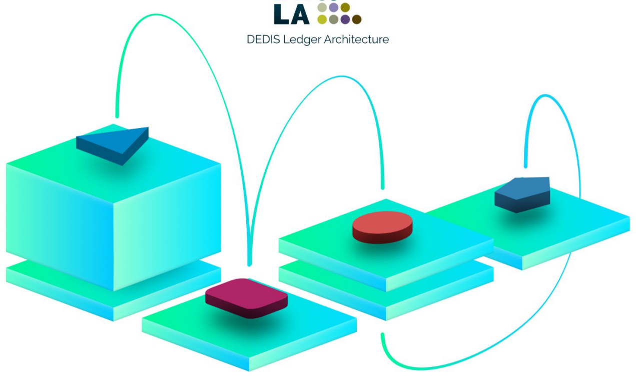


E-voting on DELA

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

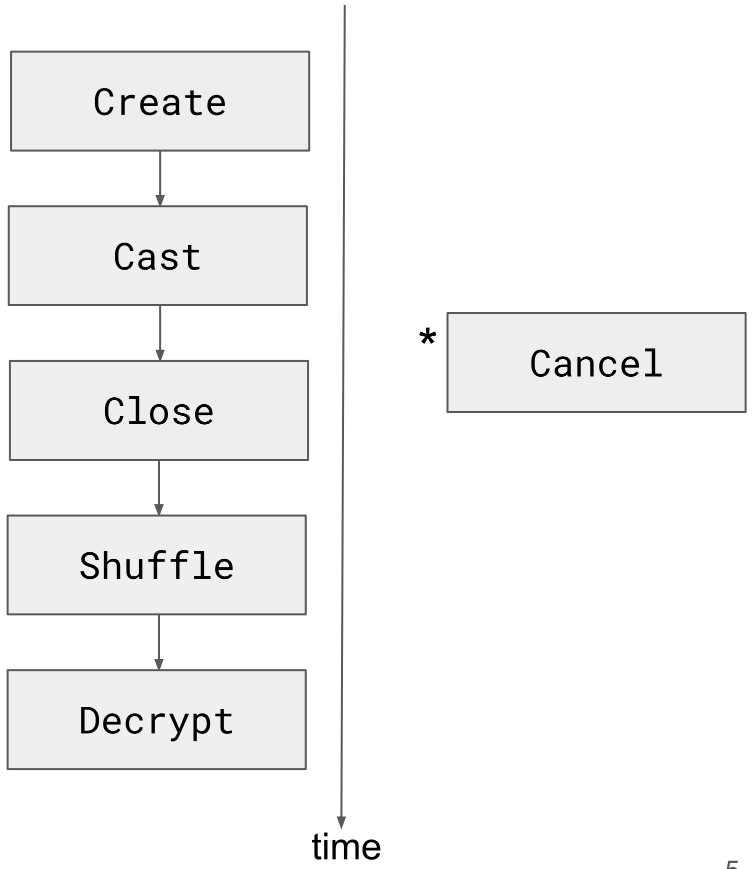
- Our goals
  - Project requirements
  - Background
- Main changes
- Evaluation
- Future work

Project requirements

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

Roles:
- Admin
- Voters

Execution:

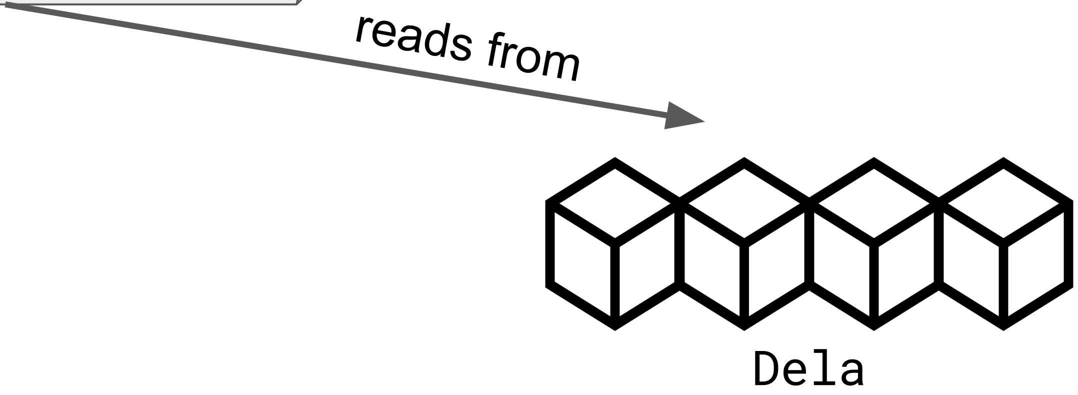


Plan

- Background
  - Project requirements
  - Background
    - Dela & smart contract
    - DKG
    - Neff Shuffle
- Main changes
- Evaluation
- Future work

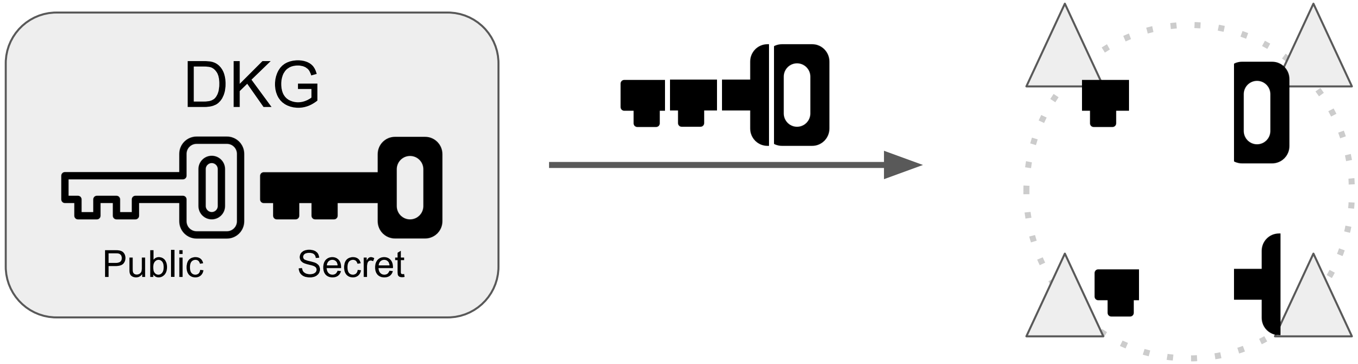
Background: Dela & Smart contract

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



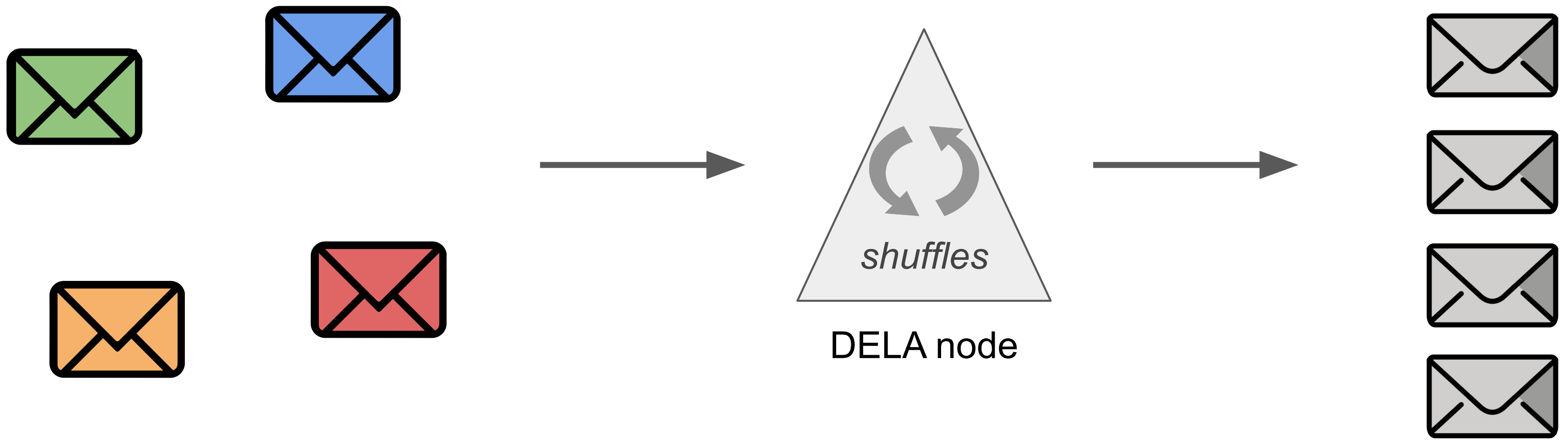
Credit: freepik.com

Background: DKG



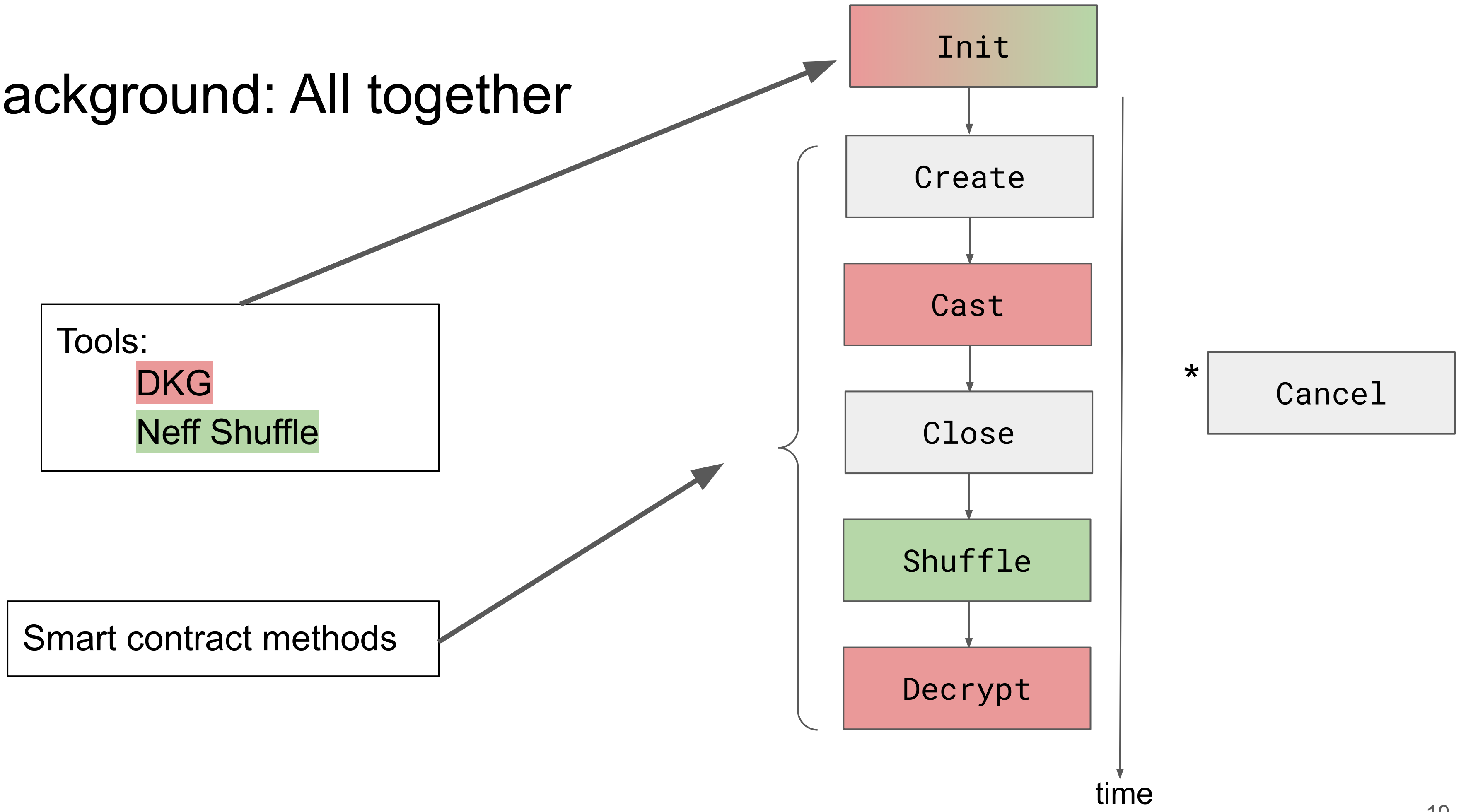
Credit: Key by Vectorstall from NounProject.com

Background: Neff Shuffle



Credit: Envelopes from favpng.com

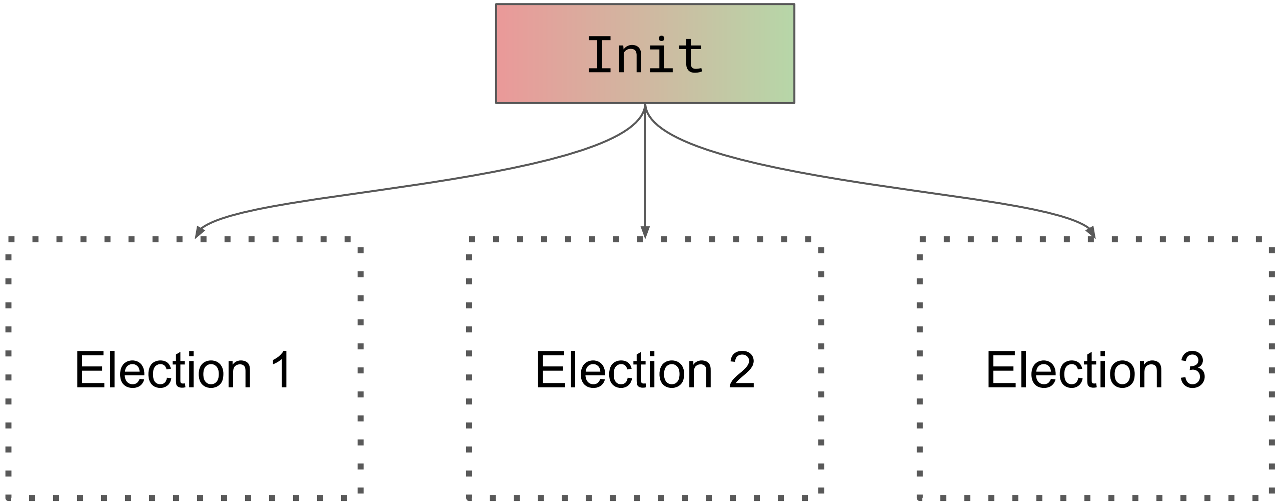
Background: All together



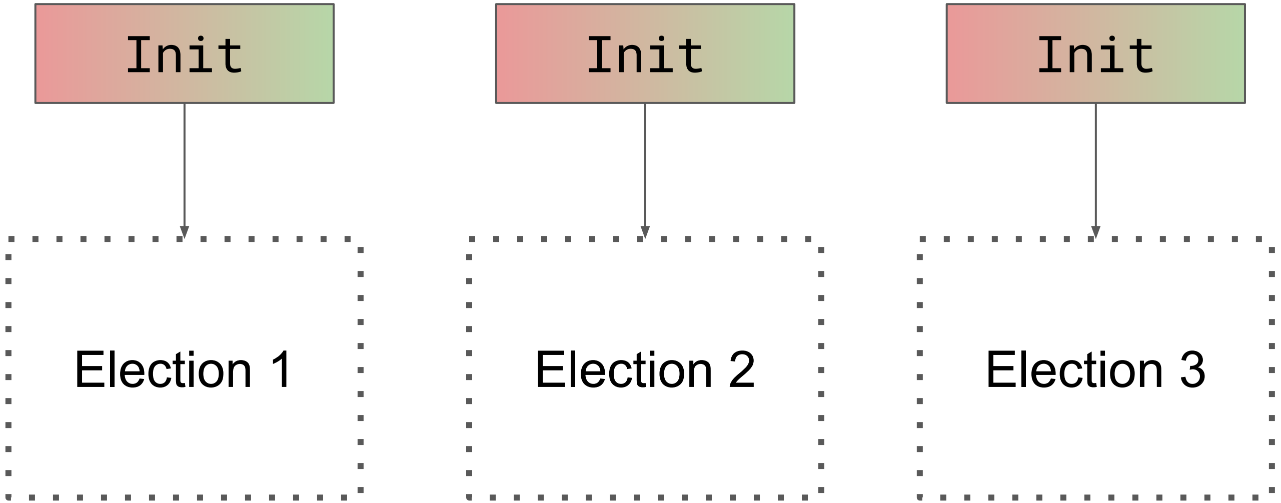
Plan

- Our goals
- Main changes
  - DKG
    - One DKG instance per election
    - Persistence of DKG credentials
  - Neff Shuffle
  - Election format
- Evaluation
- Future work

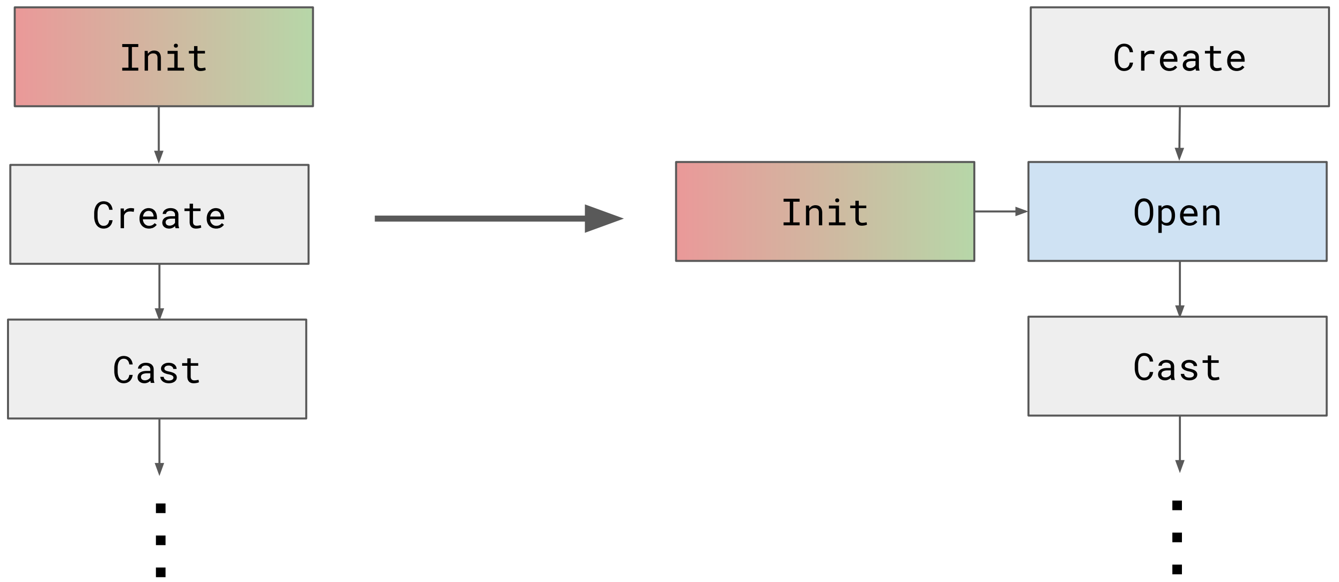
DKG: One DKG instance per election (1/3)



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
}
```

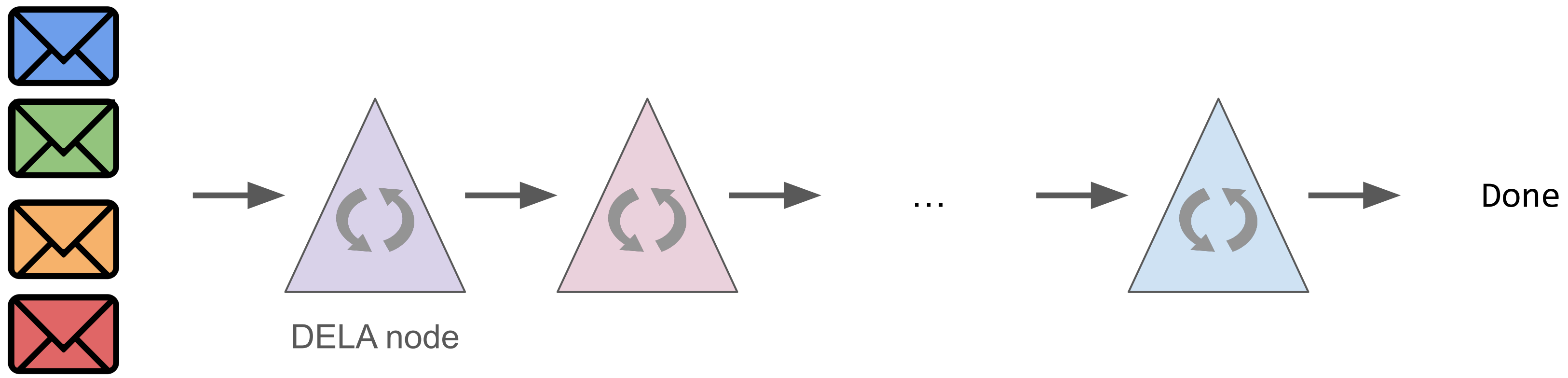
?

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

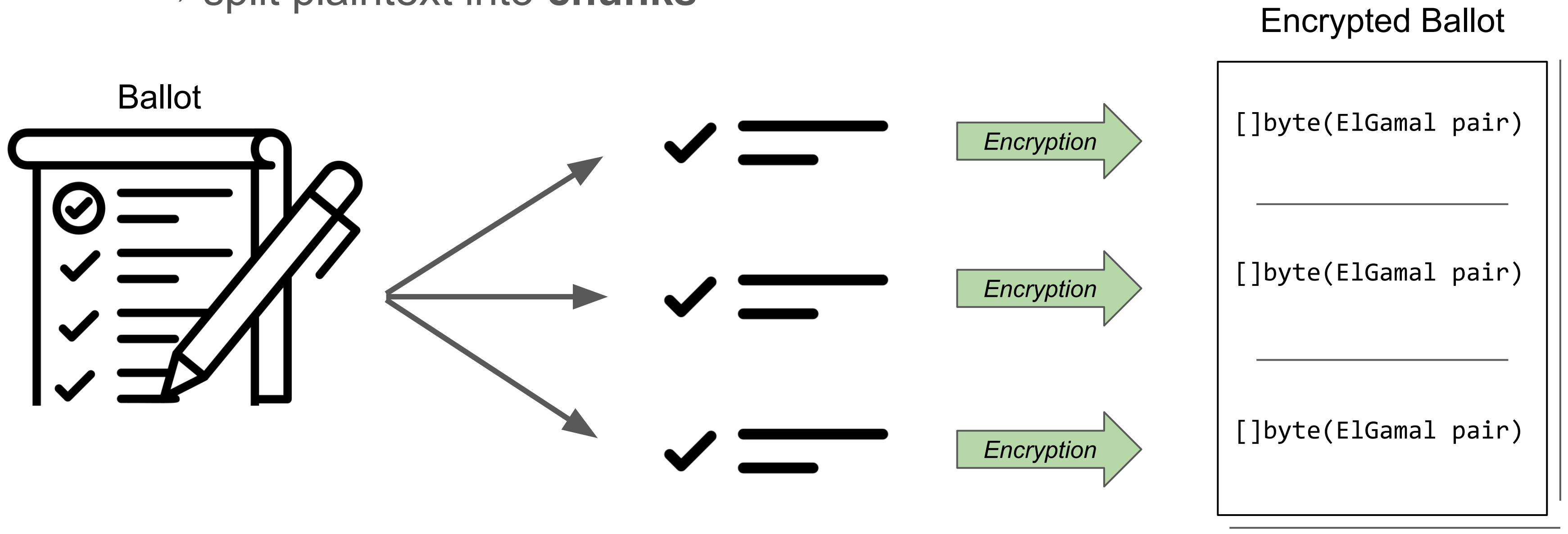
Plan

- Our goals
- Main changes
  - DKG
  - Neff Shuffle
  - Election format
    - Ballot size
    - Sequence shuffle
    - Election configuration
- Evaluation
- Future work

Election format: Ballot Size

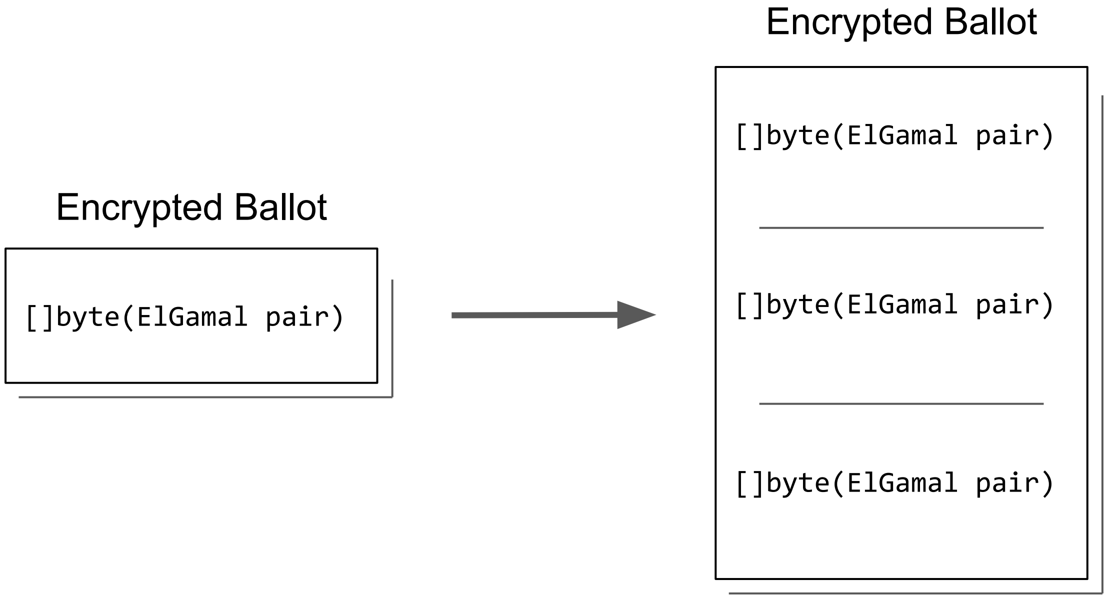
- `kyber` can only encrypt plaintexts < 29 bytes

  → split plaintext into **chunks**



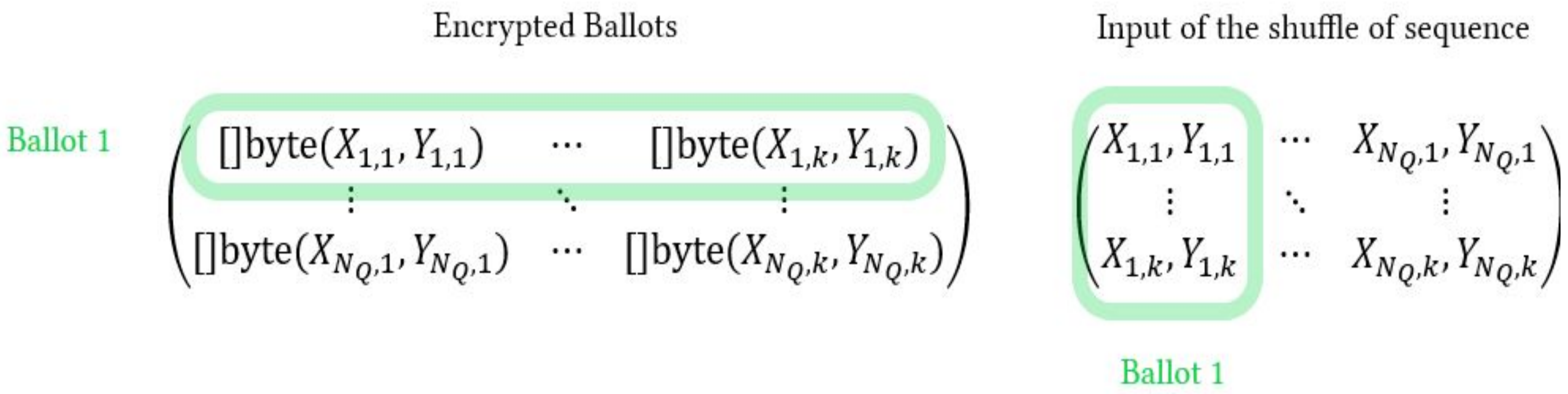
Election format: Ballot Size

- Type refactoring
- Adaptation of the protocols



Election format: Sequence shuffle

- Shuffle of ElGamal sequences
- Recent feature of kyber*


Encrypted Ballots
Input of the shuffle of sequence
Ballot 1
$\begin{pmatrix} []\text{byte}(X_{1,1}, Y_{1,1}) & \cdots & []\text{byte}(X_{1,k}, Y_{1,k}) \\ \vdots & \ddots & \vdots \\ []\text{byte}(X_{N_Q,1}, Y_{N_Q,1}) & \cdots & []\text{byte}(X_{N_Q,k}, Y_{N_Q,k}) \end{pmatrix}$
$\begin{pmatrix} X_{1,1}, Y_{1,1} & \cdots & X_{N_Q,1}, Y_{N_Q,1} \\ \vdots & \ddots & \vdots \\ X_{1,k}, Y_{1,k} & \cdots & X_{N_Q,k}, Y_{N_Q,k} \end{pmatrix}$
Ballot 1

Layout of the ElGamal pairs in memory. (X_{ij}, Y_{ij}) is the jth pair of the ith 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

```
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	

```
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

	`cothority`	last semester	now
Smart contract	37	87	76
DKG	65	90	88
Neff Shuffle	65	93	88

Test coverage evolution (%)

+ New integration tests added very recently

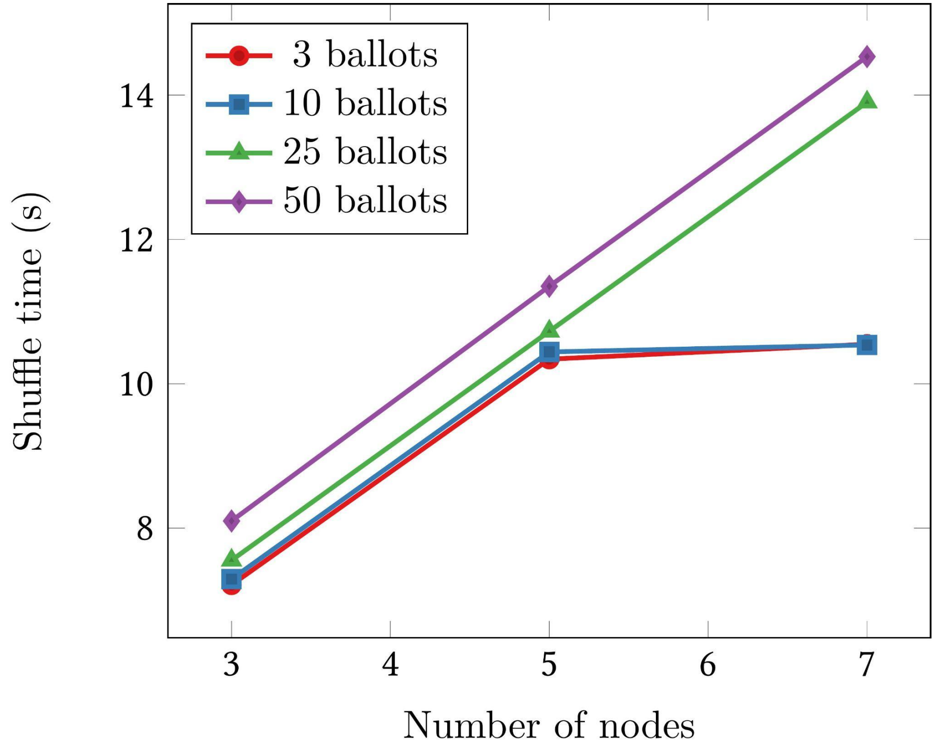
Evaluation: Performance (1/3)

- 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…

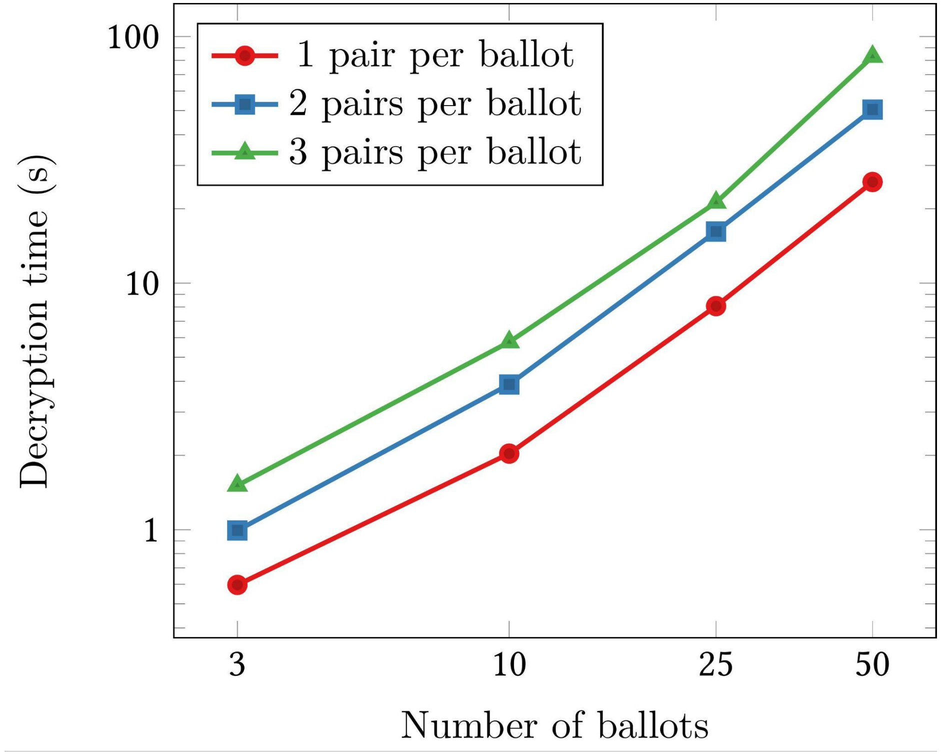
- Number of chunks is fixed at 3 per ballot



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

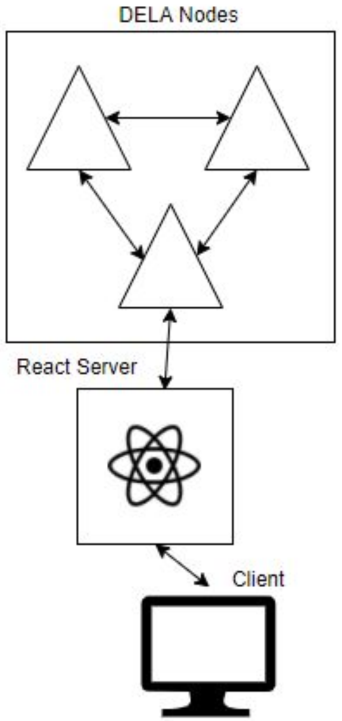
Frontend

Plan

- ❖ Tequila authentication
- ❖ Dela node request signatures
- ❖ Administration panel

How it was at the beginning

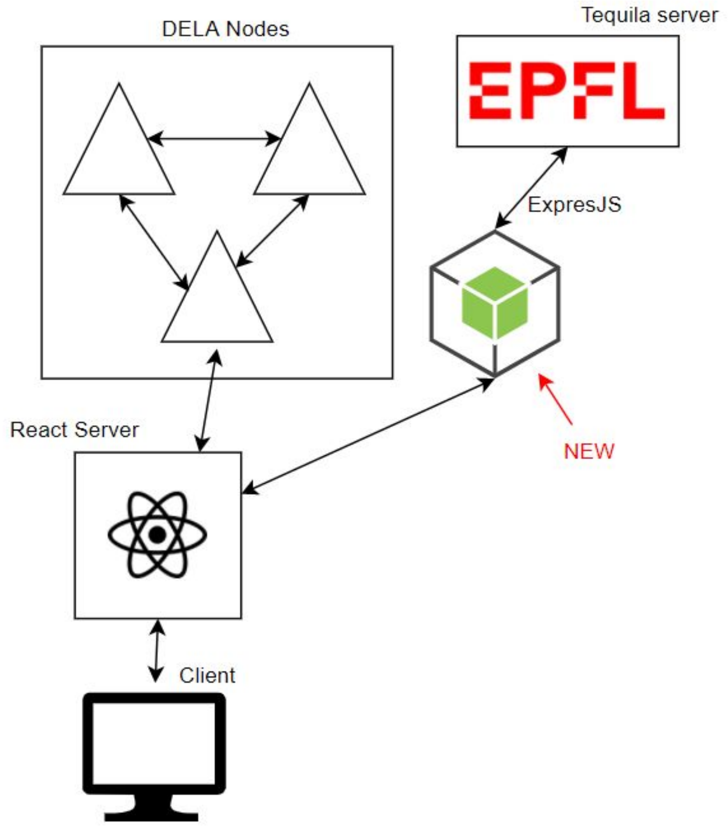
- One React process
- Anyone can login (and have a random ID) and create/manage elections and vote



Starting architecture

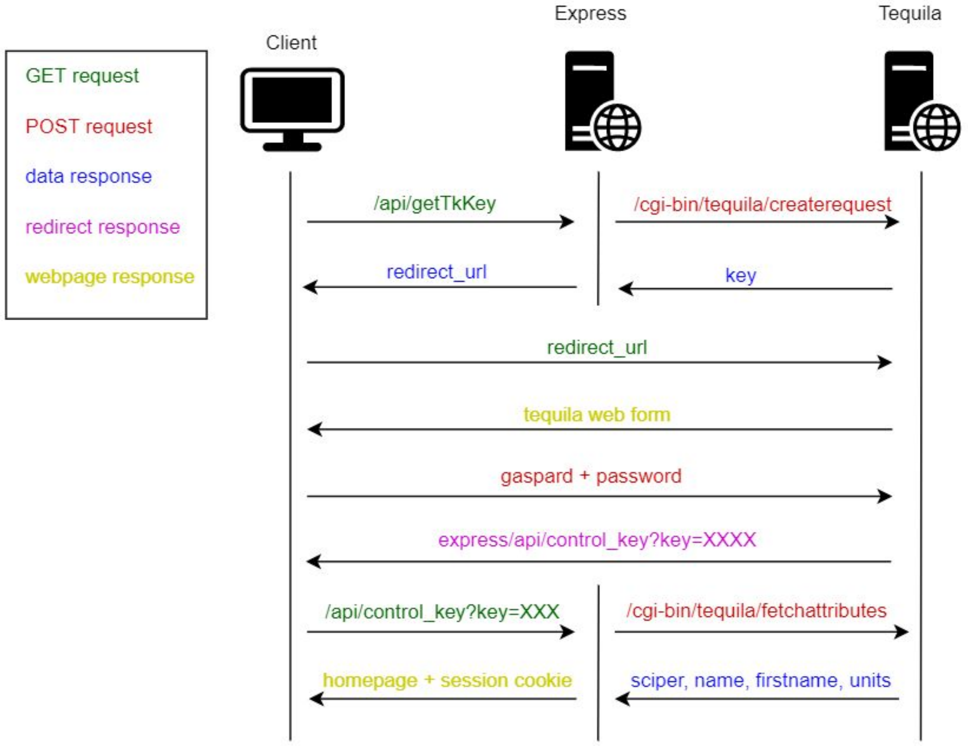
Tequila authentication

- 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



New ExpressJS backend

Authentication process



DELA message signature : reason

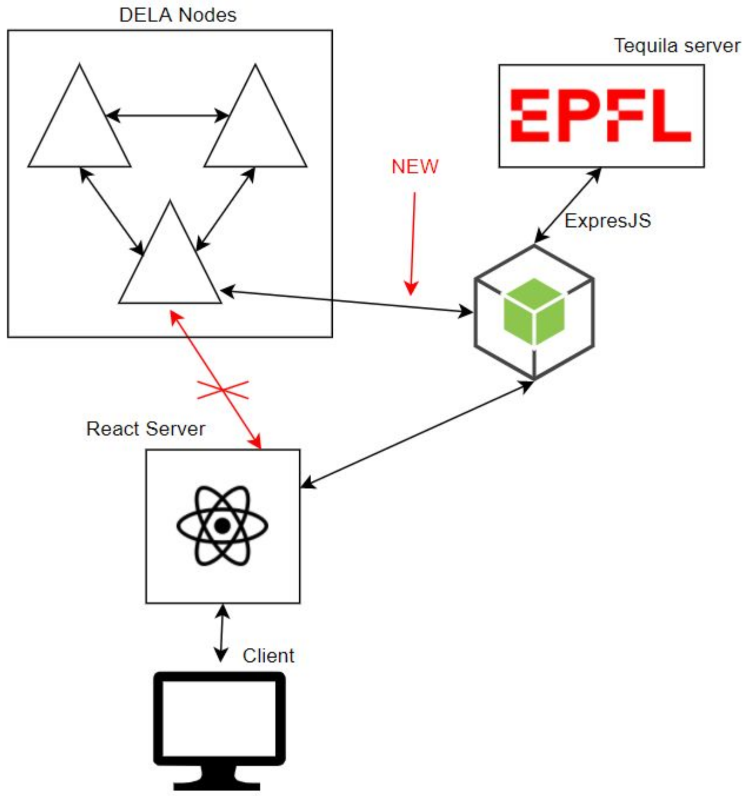
- Until now, everybody could send actions to the blockchain
- With Tequila implemented, actions must be now trusted / signed

→ Let all the traffic that goes to the DELA nodes pass through the Express

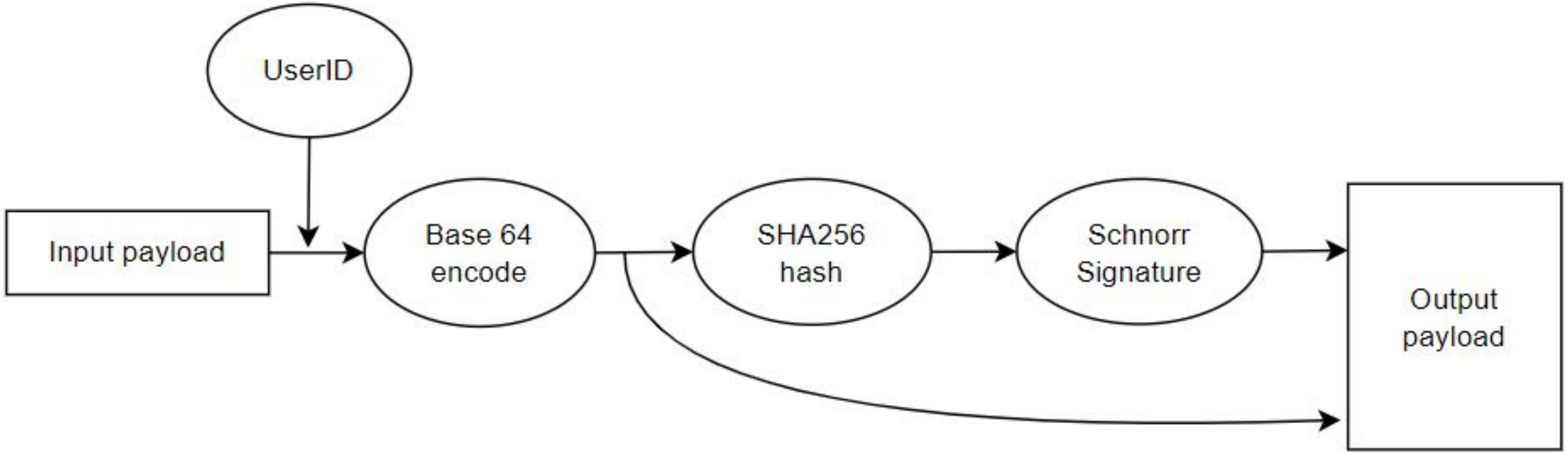
→ The Express adds current user data and signs the data

DELA message signature : architecture change

- The ExpressJS receives all requests that needs to go to the DELA nodes



DELA message signature : signing process

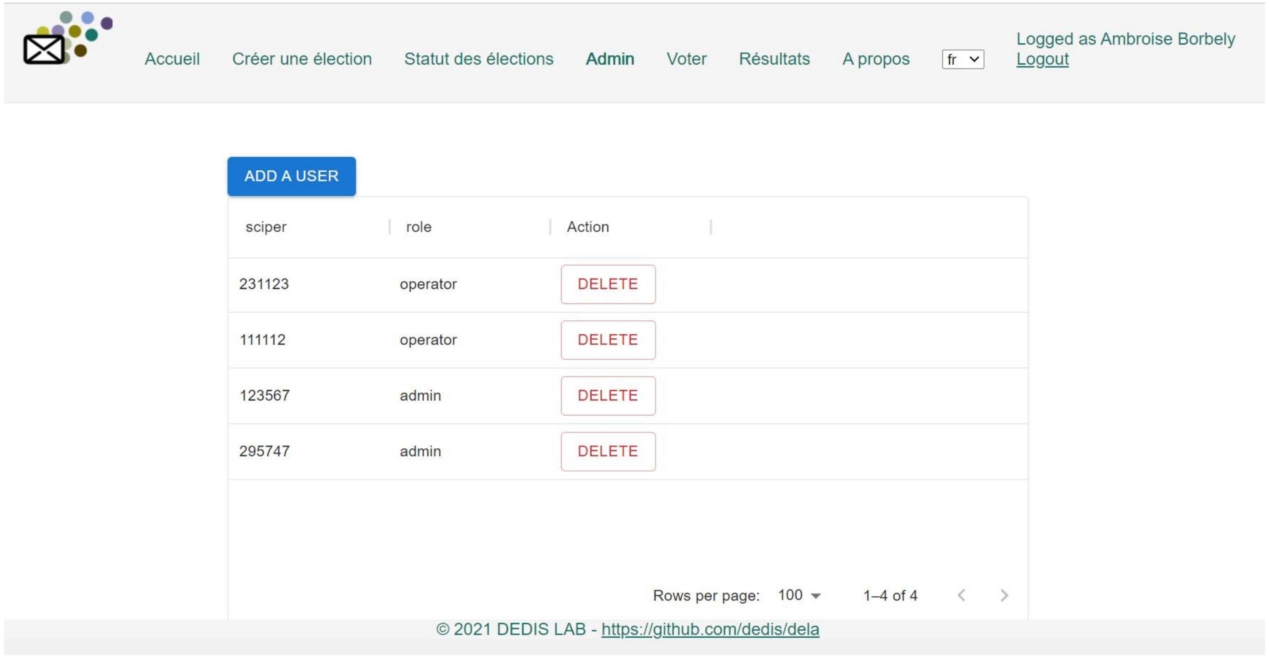


Administration Panel

- 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



sciper	role	Action
231123	operator	DELETE
111112	operator	DELETE
123567	admin	DELETE
295747	admin	DELETE

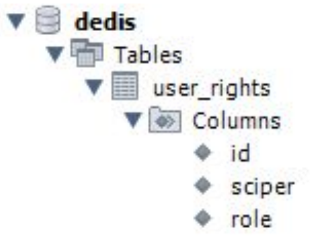
Rows per page: 100 1–4 of 4

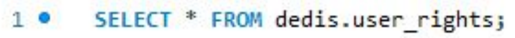
© 2021 DEDIS LAB - https://github.com/dedis/dela

View as an admin user

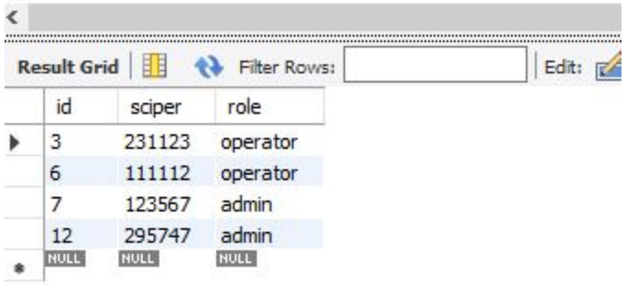
Administration Panel: Database

- Added a database with only one table to store roles

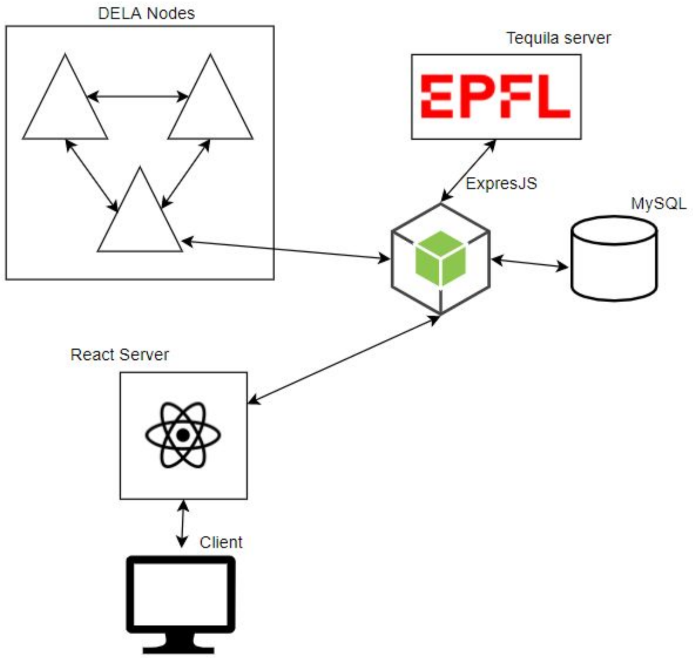




```
SELECT * FROM dedis.user_rights;
```

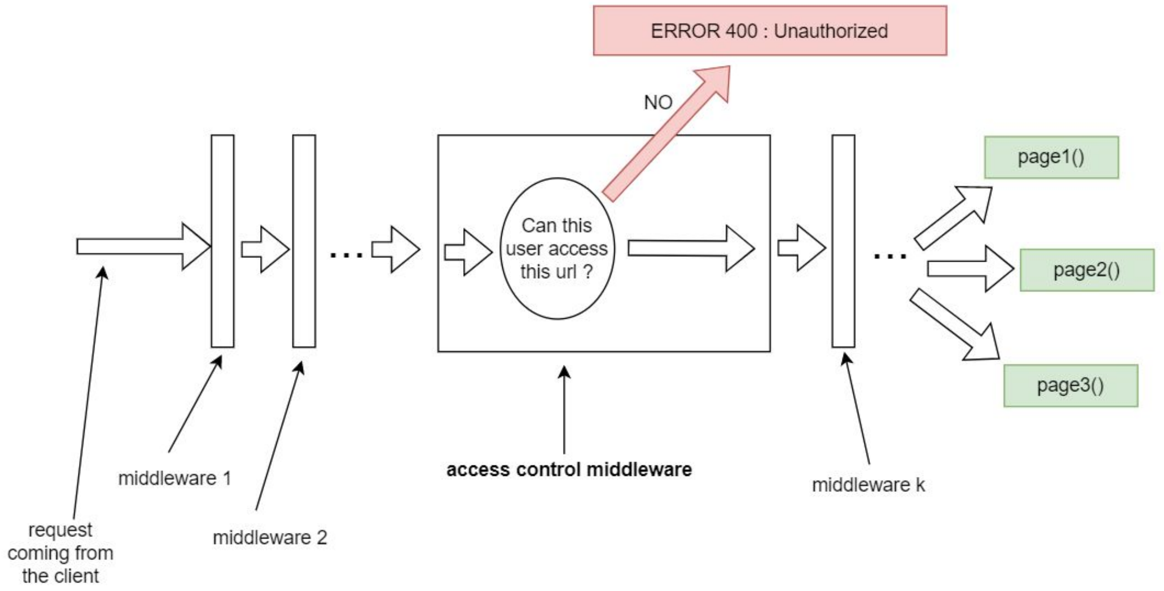


id	sciper	role
3	231123	operator
6	111112	operator
7	123567	admin
12	295747	admin
NULL	NULL	NULL



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


ERROR 400 : Unauthorized
NO
Can this user access this url ?
page1()
page2()
page3()
middleware 1
middleware 2
access control middleware
middleware k
request coming from the client

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 differents processes
  - `crond` configuration to restart the apps often

Demo

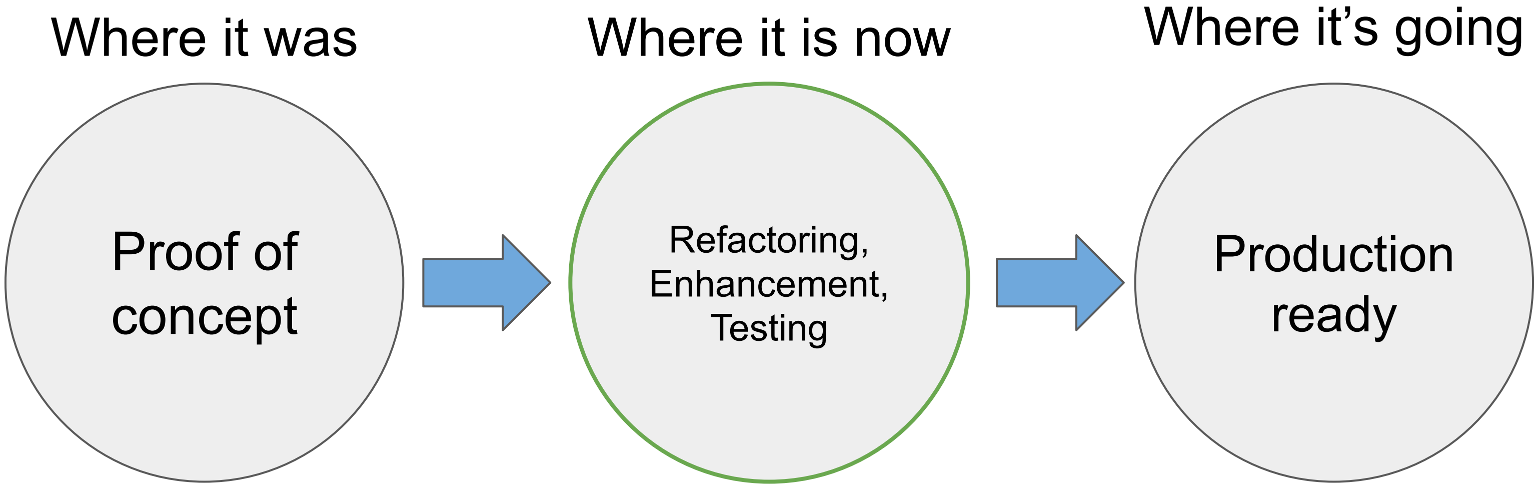
Conclusion

- 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
Where it is now
Where it’s going
Proof of concept
Refactoring, Enhancement, Testing
Production ready