E-Voting EPFL : Authentication and Frontend

Etienne Bonvin

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Semester Project

Autumn 2017

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Linus Gasser
EPFL / DEDIS

Acknowledgements

First of all, I would like to express my gratitude towards Prof. B. Ford and the Decentralized and Distributed Systems (DEDIS) lab for their confidence in letting me be part of this project. In particular, I would like to thank Dr. L. Gasser and Dr. P. Jovanovic whose help was enriching through this experience.

I also acknowledge A. Caforio who worked on the project parallel of this one and with which developing the system was an interesting practice.

Finally, I would like to thank D. Quatravaux from EPFL's STI-IT team for his advises and explanations on the authentication systems of the EPFL.

Contents

List of Figures

1 Introduction and challenges

1.1 Introduction

The EPFL is the theater of a lot of elections, some of them are lacking of participants, their unavailability being caused by distance or planning conflicts.

Inspired by an already existing voting system, Helios, we present the new online voting system, also called evoting, specific to the EPFL. It enables those previously unavailable voters to let their voices heard wherever they are and on a longer period of time by providing a simple voting interface, easily accessible and providing privacy, integrity as well as authenticity.

This project is developed in parallel of a master thesis, focusing on the implementation of the encryption and shuffle algorithms on the cothority and its skipchain while in this paper our aim is to explain the behavior and the security of both the authentication server and the frontend with which the user will interact.

1.2 Challenges

A compromise between integrity and privacy is often put on the table when talking about evoting system, ensuring both of them at the same time is kind of a challenge. A user would like to verify that his vote has been well cast and encrypted (integrity) but at the same time he should be the only one to be able to see the content of his vote (privacy).

Here we propose a solution to satisfy both with skipchains to ensure unalterable data coupled with ElGamal encryption and Neff shuffle for the privacy and the verification of the vote. Our objective being to create a truly secure system, it has to provide three key points :

- It has to be **private**, nobody should be able to see the vote of any voters. Anyone is able to know who voted but at any point what they voted for.
- It should also ensure **authenticity**, the voters can only vote using their GASPAR accounts (EPFL's authentication system), nobody should successfully pretend to be someone else and vote in his name.
- The data of the evoting system have to be **reliable**, the voter should have the certification that is registered vote is indeed what he wished to vote for.

2 Background

2.1 Helios voting system

Hosted on the address http://heliosvoting.org, Helios[1, 2] is a open-audit voting system. It is the first of his kind based on **”ballot casting assurance”**, where one voter is certified that his vote has been well taken into account and **”universal reliability”**, where any observer can verify that all captured vote were properly tallied, even though all auditors can be corrupted.

Helios solved the compromise between integrity and privacy by providing full integrity and delegating privacy to one trusted server : the Helios server. If you trust Helios’ server, you are guaranteed to reach privacy.

2.2 Schnorr signature

In the year 1990, C.P. Schnorr presented an efficient identification scheme and a related signature scheme[3] based on discrete logarithms. From then, it is considered the simplest digital signature scheme to be provably secure in a deterministic model. Its simplicity and the short length of the produced signature, around 212 bits, makes it perfectly suitable for the signatures of our authentication server.

2.3 ElGamal encryption system

Based on the Diffie-Hellman/Merkle (DHM) key exchange and described by Taher ElGamal in 1985, the ElGamal encryption system[4] is an asymmetric key encryption algorithm. It provides an additional layer of security comparing to the DHM by asymmetrically encrypting the produced symmetric keys.

2.4 Neff shuffle

The Neff Shuffle[5, 6], imagined by C. Andrew Neff is a solution of the following problem in the case of the evoting. When one is submitting his vote, no one should be able to know what he voted, meaning that after the shuffle of the ballot, no connection should be possible from the shuffled ballot to the original ballot. Still the user would like to be sure that his vote has been well encrypted and counted, hence the shuffle keeps track of the ballot for the voter to later verify his vote.

2.5 Cothority, blockchains and skipchains

A blockchain, or distributed ledger, is a log maintained collectively by a distributed group of participants, the cothority[7], who agree on and record transactions without relying for security on any single trusted party. This system allows us to trust the majority rather than a single server, increasing the reliability of the application.

The use of skipchains[8] instead of basic blockchains allows the chain to be traversable efficiently forward and backward in time, allowing us to collect efficiently the ballots in our evoting system.

3 Design and implementation

3.1 Authentication server

The authentication server is the gate between the user and the cothority. On the following graph are represented the communications between the different actors of the authentication. It ensures that a user belongs to the EPFL by contacting EPFL's authentication server : Tequila[9]. It then gives to the user a proof of his authentication to show to the cothority. Each stages is described in the following subsections, (#n) indicating that the statements correspond to the communication number n on the graph.

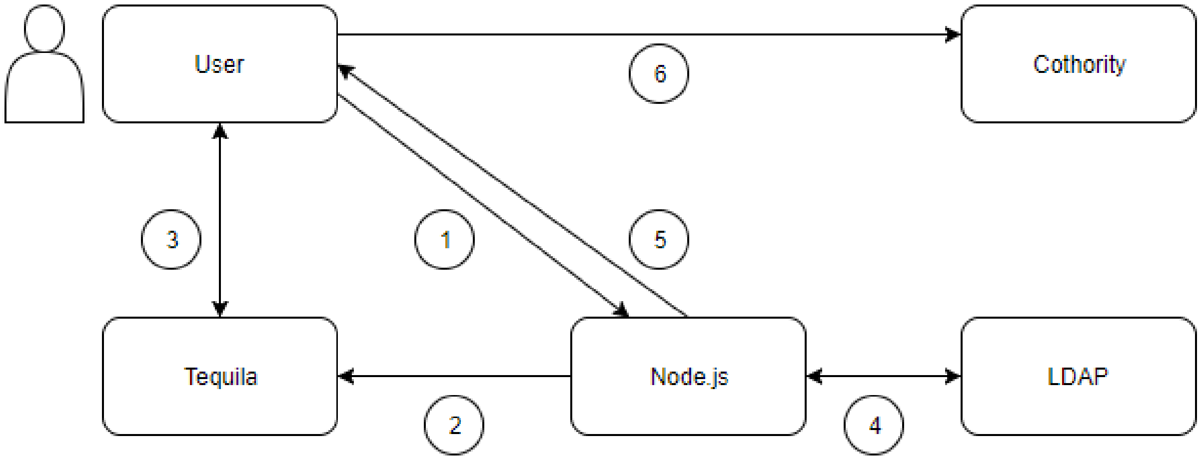

Figure 3.1: Communication overview

3.1.1 Tequila authentication

When a user asks to be authenticated, he is redirected on the authentication server (#1), which address is stored in the configuration file. The server is configured to use passportjs[10] in an Express[11] session with the passport-tequila module[12], allowing it to connect to the Tequila server and request for an authentication (#2).

The tequila servers then contact directly the client for the rest of the process, it returns to the authentication server when the Tequila login is finished with the connecting user's informations (#3).

3.1.2 LDAP request

After the authentication with Tequila is successful, a LDAP request is initiated to recover more informations about the user (#4).

```
var opts = {
  filter: '(&(objectClass=person)(cn='+name+'))',
  scope: 'sub',
  attributes: ['uniqueIdentifier', 'memberOf', 'dn']
};
```

```
client.search('o=Ecole polytechnique federale de Lausanne(EPFL), c=CH', opts,
    function(err, res) {
    // ... Correctly parse the result
});
```

The filter attribute indicates what we are looking for, in this case we are looking for a person with the name 'name', this variable being initialized at the end of the Tequila authentication with the name of the user returned by the Tequila server.

Then we say that for the filtered people we want to get the following attributes :

- dn to get a set of basic informations, the one interesting us here being the section.
- uniqueIdentifier for the sciper (user's unique identifier at EPFL).
- memberOf to get the EPFL groups the user belongs to.

3.1.3 Signature

Once the informations are recovered using the LDAP request, a message is created and signed using the Schnorr algorithm using the private key stored in the configuration file.

```
var signed_message = {
           id : result.sciper
       }
var signature = schnorrSign.signMessage(signed_message);
```

This message is then given along with the signature to the user (#5). It proves to the cothority that he has well been authenticated by the authentication server through Tequila (#6). The message only contains the sciper of the user since it is the only user's information used by the cothority for now but it induces some security problems that are developed later in *section 4.2*.

Sections and groups in which the user belongs are not needed yet in the rest of the application, however everything is already ready so that they can be added to the message to filter the elections a user is allowed to vote in. The following part of code is the message containing all the recovered informations about the user that can be used in later versions of the application :

```
var message = {
    id : result.sciper,
    name : req.user.displayName,
    /* The different representation of the section of the user. */
    section : result.section,
    /* The EPFL groups in which the user belongs. */
    groups : result.groups,
    fullDate : new Date().toLocaleString()
}
```

Note that this full version of the message is much more secure than the previous one, by signing the time stamp on which the authentication has been made. With such a message, we would be able to avoid repetition attacks.

3.1.4 HTTPS

Since November 2017, each communications with the Tequila servers should be done only by certificated servers using HTTPS connections. A certificate will be needed at the website's deployment in order for the applications to be able to communicate with Tequila servers and provide user's authentication, this can be requested through EPFL's registration authority available online at https://rauth.epfl.ch.

3.2 Communication cothority / frontend

After being authenticated with the Node server (#5), the user will have to show the message and the generated signature to the cothority composed of several servers that we will call equivalently conodes. For this purpose, a communication is established between the user and the conodes to allow the user to log in and then create, manage or vote in elections.

To ensure the communication between the cothority and the frontend, protocol buffers have been used, protobufjs[14] on the frontend side. The communication is initiated the following way :

- The IP address of the node we are sending the message to is stored in a configuration file.

```
const node = {
    Address: 'tcp://'+nodesIp+':7002'
}
```

- The socket is created using the address of the node and the messages of the protocol.

```
socket = new dedis.net.Socket(node, messages);
```

- A message can then be send by simply using :

```
socket.send('Query', 'Reply', dataToSend).then((receivedData) => {
  //Use received data
}).catch((err) => {
  //Manage the error, most of the time just display it.
});
```

  The variables name of this example are self explanatory, we send a message 'Query', and then wait for a message 'Answer'. 'receivedData' is our promised data on which we will execute a specific behavior and we stop the execution on error.

Now we have to define a communication protocol which will describe the format of the exchanged messages.

3.2.1 Communication protocol

Let's have a look at the structures of the protocol as well as at the Message / Answer pairs during each steps of the voting process and explain their usage.

Structures

In order to communicate efficiently, a set of structures have been defined representing the different objects of the election.

- **Election** : containing a set of useful data defining the election such as the name, the description, the deadline, the creator, the participants and the voters.
- **Ballot** : containing the name of the voter who submitted the ballot, has well as the encrypted choice in an [alpha, beta] ElGamal pair. A field for the plain text is also available for decrypted ballots.
- **Box** : containing only an array of ballots, result of the aggregations.

Messages

Now that we have some base objects to communicate, we define messages and reply pairs for the actual communication. Figures 3.2 and 3.3 at the end of the section represents those communications and their impact on the stored data.

- **Login / LoginReply :** This is the first pair of message exchanged during the process. The user request a login by presenting his sciper and the authentication server's signature and the cothority either reject the authentication or accept it and give a level of administration to the user as well as a session token that the user will be able to use in the other message to prove his authentication.

- **Open / OpenReply :** An administrator has the possibility to create, to open, a new election. For this, he presents to the cothority the election he wants to create. Later, the created election will be referenced by the ID of the skipchain in which it has been stored.

- **Cast / CastReply :** When a user wants to cast a ballot for an election, he sends to the cothority a message containing the election in which he wants to vote and his encrypted ballot.

- **Shuffle / ShuffleReply :** Once the deadline of the election is reached, the shuffle of the ballots can be initiated by the creator of the election. For this he sends a Shuffle message with his session token and the ID of the genesis block of the election skipchain. The shuffle of an election can take some time. When it is finished the cothority answers with the box containing the shuffled ballots.

- **Decrypt / DecryptReply :** When the election is finished and shuffled, the ballots can be decrypted. This decryption is also initiated by the creator of the election with the same informations as for the shuffle. Once finished with the decryption, the cothority gives a box of the decrypted ballots to the user.

- **Finalize / FinalizeReply :** The finalize message is a combination of the shuffle and the decryption message. However in order to make the code clearer, it is not used yet.

- **Aggregate / AggregateReply :** The different aggregate messages allows the users and the admins to check the result and verify the different steps of the election. The aggregate message has three different types depending on the type of recovery desired : the encrypted ballots, the shuffled once or the decrypted ballots.

3.3 Frontend

This part describes the structure of the user interface and the way any user can display, create, vote and finally manage various elections. We also have a look at how the different stages of the election can be verified by the voters and the creator of an election.

3.3.1 Single web page application

One of the key point of the frontend is to provide a single page web application. In this type of applications, rewriting the current page is preferred as loading an entire new page from the server. The changes are all made through jQuery. In that case, no informations about the user's vote are sent across the network before their encryption, when the ballot is cast.

3.3.2 Conventions

Date format

A convention was needed to enter the date for it to be well interpreted and to avoid mistakes. Hence the deadline should respect the following rules :

- It should be in a format DD/MM/YYYY.
- It should be a valid date.
- It should be a date equal or later than the day after the creation.

Election status

An election goes through different stages from its creation to the decryption of the ballots :

- **on voting** : the deadline of the election has been reached, the users can still vote and no other actions are possible from the admin.
- **finished** : the election is finished but not shuffled yet, the users can't access any results they have to wait for the admin to shuffle the election.
- **finished - shuffled** : the election is finished and shuffled but not decrypted yet, still no results or aggregations are available.
- **finished - decrypted** : the election is finished and decrypted, the users and the creator of the election can now aggregate the ballots at the different stages of the election : before the shuffle, after the shuffle, after the decryption.

Election data

The election object has a special data field meant to contain any relevant informations. Here we decide to put in the participants, i.e. the scipers for which we can vote. Each participant sciper is then translated in 3 bytes and stored one after the other in the data array. The length of this array is then divisible by 3.

Ballots display

Each stages of the election will be displayed differently. Since the election is an open-audit one, a user and an admin have access to the same resources. The only difference is that a user is, for each stage, displayed his own ballot at that stage so that he is able to verify the integrity of his vote.

The stages are displayed in a grid[15] showing the relevant and allowed informations.

For the **voting** stage, we display the ballots with the sciper of the voter and the ElGamal encryption pair. The sciper is displayed in plain since the important thing to hide is not who voted but rather what they voted.

For the **shuffled** stage, we now only display the shuffled ElGamal encryption pairs without the associated scipers. Since this stage is meant to lose track of the ballots, only the voter who cast the vote is able to see which shuffled ballot is his own, but none of the other voters.

Finally, on the **decryption** stage, the results are displayed with the participants ordered in ascending order regarding their number of votes, this number being also displayed. A voter also sees his vote clearly displayed, again to verify the integrity of his cast vote.

3.3.3 Design and Structure

Two different web pages have been created for the two different actors of the evoting system :

- **The user page :** every people belonging to the EPFL can access this page, it allows the user to vote in any elections he has been declared as a voter.
- **The admin page :** only people with a special access (listed when creating the master skipchain) can access this page. On this page an admin user is able to create an election and manage, that is shuffle and decrypt, the ones he already created. It is not possible to vote from the admin page.

3.3.4 Admin experience and usability

Login

When reaching the administrator page, the user is proposed to log in, he is then redirected to the tequila servers and invited to enter his EPFL's credentials.

After successful authentication, the signed message from the authentication server will be sent to the cothority which will verify the validity of the signature as well as the administration level of the logging user. The user has to be an administrator to the cothority to access any further steps.

If the authentication with the cothority is successful, the admin user can then either create a new election, either manage an election he already created or finally log out.

Create elections

By clicking on the 'Create election' link on the navigation bar, the election creation screen is displayed.

Here he is invited to enter the details of the election : the name, the description, the end date, the list of the participants and the list of the voters he will be able to verify the entered informations and change them if necessary before validating the creation of the election.

Manage elections

When he is on the election list page, either by clicking on the link in the navigation bar, either after logging in, the admin have access to all the elections he created.

When clicking on a list item, the admin can see the details of the election as well as the election status and the actions he is able to perform at this moment depending on the status of the election.

- **on voting :** the admin can't do anything, he has to wait for the deadline to be reached.
- **finished :** the admin can initiate the shuffle. Once the shuffle is finished, the election changes status to finished - shuffled. This step is really where the privacy takes place, no one will be able to come back from the shuffled ballots to the initial ones but the one who cast the ballot himself.
- **finished - shuffled :** the admin is invited to see the ballots at the three stages of the election : the encrypted ballots, the shuffled ballots or the decrypted ballots. The ballot will be displayed for each of those stages regarding the section *3.3.2 : Ballots display.* The first time the decryption is requested by the creator, it will launch the decryption of the ballots for everyone and changing the status of the election to finished - decrypted.

- **finished - decrypted :** the frontend will behave the same but the decryption request will not launch a new decryption and will display directly the previously decrypted ballots.

The display of the encrypted ballots and of the shuffled ones offers the opportunity to the creator of the election to verify that each ballot has been well shuffled, i.e that their ElGamal encryption have been well changed between the two phases and that there's no mean to back from the shuffled ballots to the original ones.

Logout

At any time, a user can log out of the application by clicking on the corresponding button in the navigation bar.

The logout will simply drop the session cookie and the list of elections of the admin before redirecting him to the home page, inviting him to log in.

3.3.5 User experience and usability

Login

The login process for the user is almost the same as the one for the administrator, the only difference being that there is no access restrictions, no administration level. Hence every user with a gaspar account can access this page.

After successful authentication, the user can either interact with one of the election he can vote in, either logout.

Vote in election or see result

When he is on the election list page, a user can click on one of the election list items which are the elections he is able to vote in.

On click the election details are displayed. What is displayed underneath depends on the state of the election :

- **on voting :** the user see the different choices for the election. He can click on the radio buttons near them to select one and then click 'Submit' which will encrypt the ballot using ElGamal encryption system and send the encrypted data to the cothority. The ballot will be stored in the skipchain dedicated to the election. A user can vote multiple times for the same election, however only his last vote is taken into account.
- **finished** or **finished - shuffled :** the user can't vote anymore. In this state he has to wait for the admin to shuffle and decrypt the election.
- **finished - decrypted :** the results of the election and the ballots at the different stages are now available to the user. His own ballot at each stages is also displayed so that he verify both the integrity of his vote and the privacy brought by the shuffle.

Logout

The logout process is the same than for an admin.

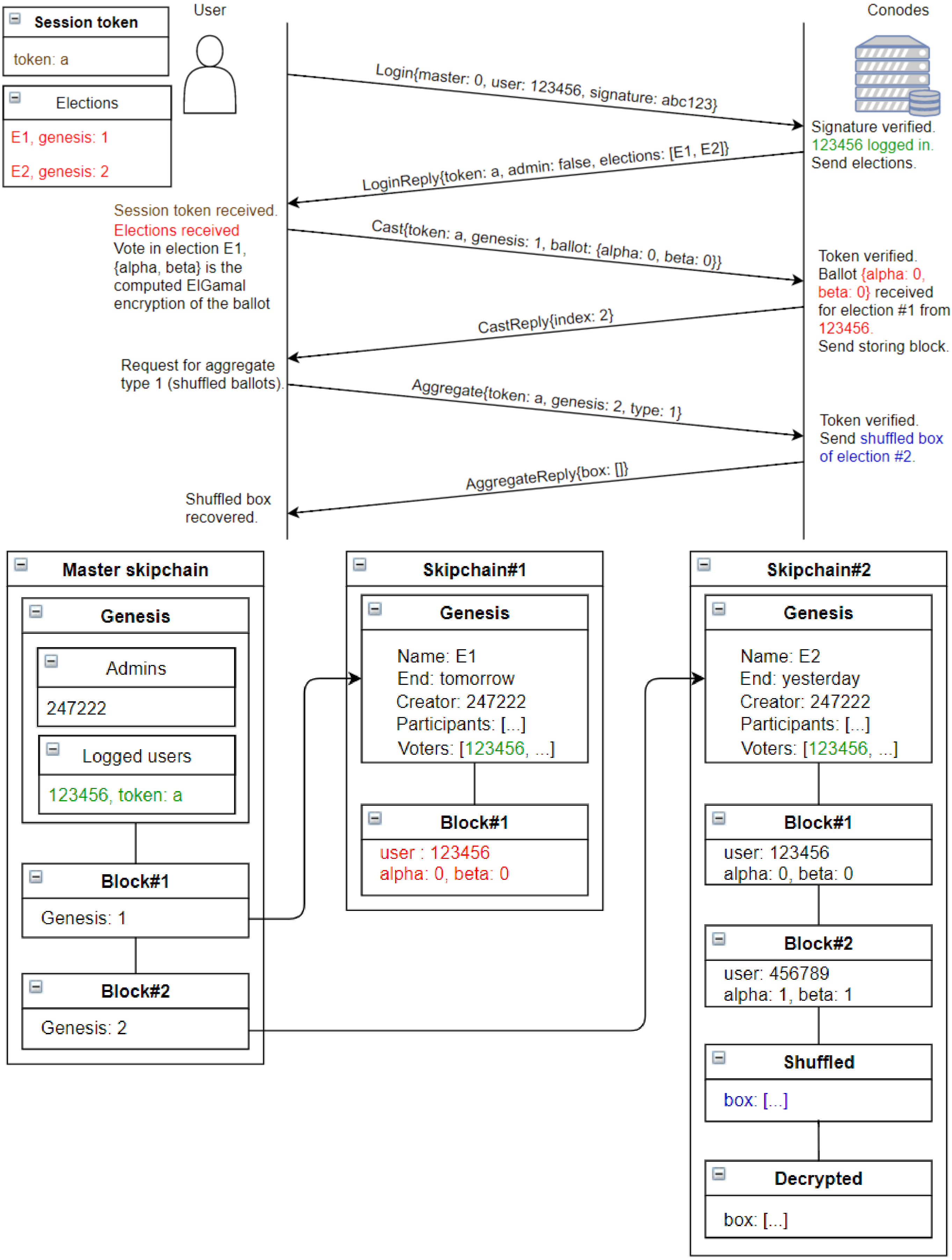

Figure 3.2: User / Cothority Communication Scheme

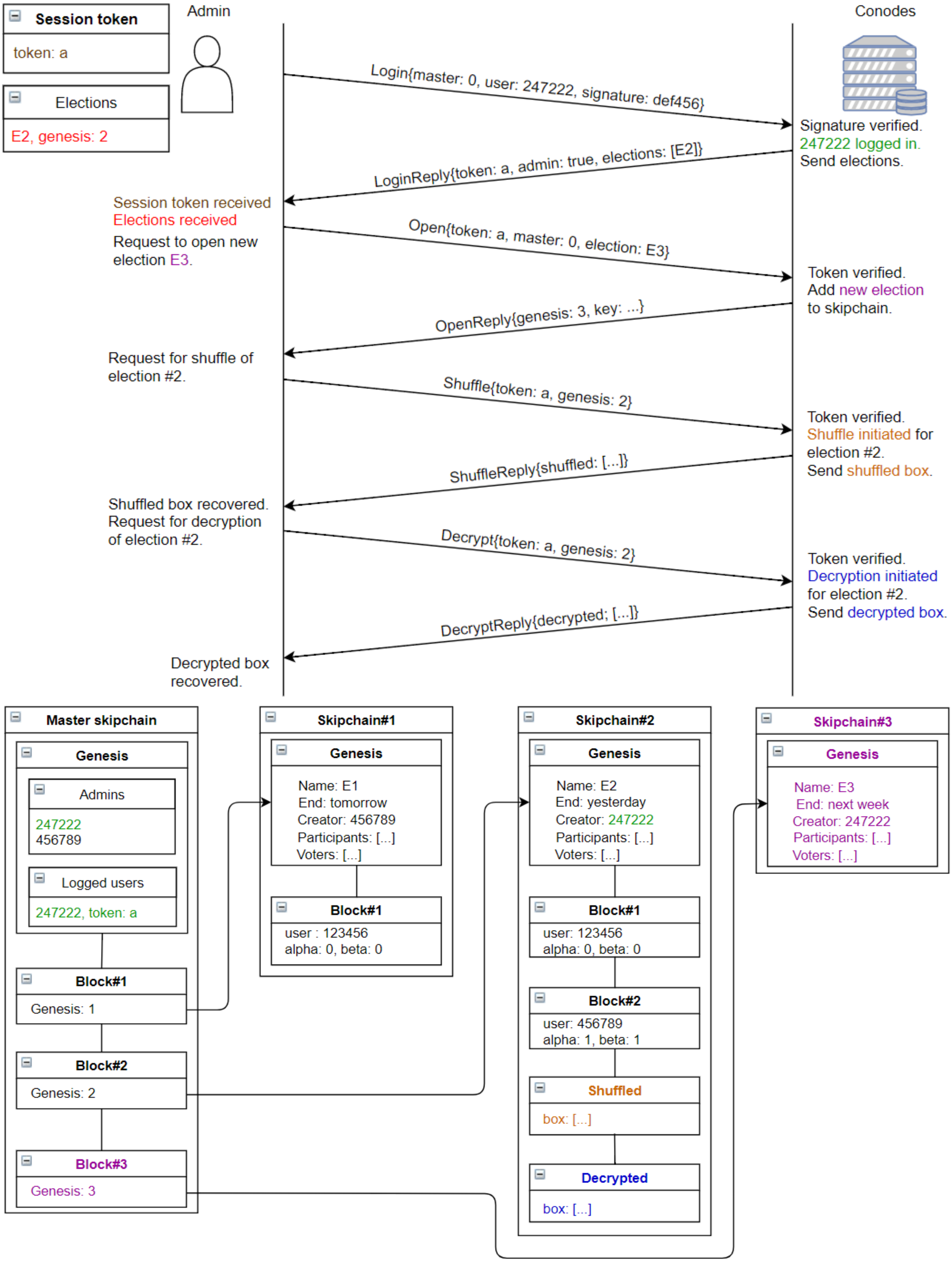

Figure 3.3: Admin / Cothority Communication Scheme

4 Theoretical and practical limitations

4.1 Nodejs server, a possible weak link

In this part we will see in which extends the choice of a nodejs server as the authentication server can be considered as a weak link of the system.

4.1.1 Loosing the no trusted server idea

Recall that the basic idea on which a cothority is built is to not trust any server in particular but rather the majority of them.

Hence if a server is corrupted, it will not be taken into account in the operations since it would not be part of the majority. The majority of the servers needs to be corrupted in order to alter the result given by the system.

The problem of an authentication relying only on one server is that it forms a sort of contradiction to this concept, we trust the majority to count one's vote but we trust only one server to trust who voted.

As we often say, the resistance of a chain is the resistance of its weakest link, it is even true for skipchains relying on a Node.js server. Imagine an attacker would like to change the vote of a specific person, imagine he is not able to corrupt the majority of the servers of the cothority, thus not able to corrupt the person's ballot. With the actual configuration of the system, the attacker could target the authentication server, impersonate the user and make a new vote in his name, rewriting his old vote.

This problem is even bigger considering the actual signature of the authentication server who is vulnerable to repetition attacks.

We can imagine that scenario happening for each, or at least the majority, of the voters and then the result of the election would be altered.

4.1.2 Partial solution : HTTPS

Since November 2017, the Tequila servers are only accessible for servers with an HTTPS connection and then a certificate. A communication of this type is more secure than a basic HTTP connection, it avoids attacks like the Man-In-The-Middle for example.

However it does not provide a perfect security, the authentication being once again vulnerable to repetition attacks for example.

4.1.3 Better amelioration : a distributed authentication

A way better resolution of this problem would be to treat the authentication the same way the other informations are treated in the cothority, i.e. to decentralize it.

Instead of having a user connecting directly to the Tequila servers, we could have a user sending his encrypted credentials along with a time stamp and a nonce to each server of the cothority, the servers would then try to authenticate to the Tequila servers. Therefore if the majority of the servers of the cothority get a positive answer from the Tequila servers, the user would be considered successfully logged in, otherwise his authentication would be rejected.

One can easily see that such a solution will be more suitable considering the design and the general idea behind the cothority.

4.2 Authentication message vulnerability

In its actual version, the authentication message described in *section 3.1.3* is very unsecured. Recall that the only information used is the sciper of the user, signed by the Node.js server. Even though we assume that the Node server is the only one able to generate the correct signature and that the communication is encrypted, the process is not entirely secured. We describe here the main vulnerability, which is against **repetition attacks**.

4.2.1 The attack

Assume that an eavesdropper called Eve is listening to the communications between Alice, the user, and the conodes. Alice goes through the authentication process, she connects to the Node server, identify herself successfully with Tequila and get a message back with her sciper number and the associated signature. She then contacts the conodes, encrypts a Login message - as described in *section 3.3.5* - using the conodes' private key and send it. Eve, listening to the communication, sees this message, even if she can't know its content because of the encryption, she knows that it corresponds to Alice's Login message since it is the first message sent between Alice and the conodes in the protocol.

On a later occurrence, once Alice is disconnected, Eve will be able to send the message again, successfully pretend to be Alice, since the conodes have no ways to differentiate this message from the previous one, and then be able to vote in Alice's name, overwriting her initial vote.

4.2.2 How to do better

By adding the date at which the authentication has been done - called a timestamp - to the signature, each signature will be unique for each Login session. Hence if Alice connects at 2pm, she will send the signed message containing {date: DD/MM/YYYY, 2pm}. The conodes verifies the signature and if the timestamp is still valid, i.e that the date in the message is not a date too far away in the past.

Now if Eve sends the same message with the same signature, say she sends it a 3pm, the signature will be successfully verified but the timestamp would not be valid and then Eve would not be able to impersonate Alice.

4.3 Elections limited to the scipers

Due to some storage problems in the cothority, only the scipers could be used for the representation of the participants of the election.

Unfortunately this way of displaying the participants is not ideal. The only way to recover names from scipers is through LDAP request, the problem being that a LDAP request can be successful only when send from EPFL's WiFi or using the EPFL's VPN. We can't assume that the user will be connected through one of those, hence disabling us to use LDAP in the browser.

Another solution would've been to recover the informations through another proxy server - another Node.js server - which task would be to only recover EPFL's users informations or to reuse the authentication server for this purpose. But as we saw earlier with the problem of the authentication server, doing so would have mean to trust again one specific server for a task which breaks the idea behind the cothority.

The best solution to this problem would be to find a way to encrypt any type of data in a byte array and set it as the data field of the elections. Hence any type of vote could be represented.

5 Results

5.1 Comparison with Helios

The main difference is a change in reliability. In Helios, you have to trust the Helios server for privacy. The major problem of this being the outcome when the server would eventually get attacked. It's reliability will not be ensured anymore and neither will the privacy of the votes.

By using a cothority you do not have to put all your trust in one server but rather on a majority of servers. In this scheme, a corrupted server has no power and can not alter the election in any way.

5.2 Presentation of the results

Here we present the results of the frontend, what is displayed to the user after an election with 45 votes. The first figure represent the ballots shown to the user in a grid, with its own rewritten above, he can verify that is vote has been well taken into account by searching in the grid. The same way, the second figure show how the shuffled ballot of the user is displayed followed by the grid of all the shuffled ballots. And finally the third figure displays the decrypted vote of the user and the results of the election.

Your ballot's encryption pair :

b29c9c202d905c37c35cdc1ad6336d2e34a9778e388044c555cb5517eb1fffd1

125cc07ab786b1f841651c541fbb53528241e11e8fa377dba78a1b79b4a05d0e

All Fields Search

Sciper	Alpha	Beta
100024	[illegible]	[illegible]
100035	8a4d9e227eb13d39d310c59f2df4e42e0bfba6731d40840c45b9e0a1acf4ff5d	950f133428cce50d9af843a44629b1d6f16bc8bd359d27584fa81711a2ac2dc4
100019	a0815857d633627f4a5fec3baa075039fe2b378447ab8be2d8d6f9bba63b6292	e720f1f894fa2e41f88f2a05adcfcded28e2665846904cc79465f66912d6a0c9
100025	970c2e225c824d3a4c1013cf28af75a759b835c1b479957f69ca42840c2f9ebb	9614ce52dae534b108078ec6336a1ee6a0738a2f4b1f74008825a54501ee0674
100042	2cf5fc7c6c06e1735e75862698e874338ee9765dbbcea12e7ff9dc5f156735b3	69eafd4fee824a04d1c31d3e0ce89aef1ed3c75ea008b4fcbf08beaedda979a9
100043	0d49dabcfb79bb5fc7cf491f13e2cb1d99e4ae33598aa74729d99626cd333678	785485e36acf66bbb240a7c49380f9dbfb26c9792c663bf94068c0225e7a831d
100048	8f4e2cf999cb73140fadc1d1ef29bcc9ec2e552a21b0c91aa338058ea7cee982	c90e5a2ae8449877afb7452ba78956a7aa18818cc5d46a81d9c2e23adbcd408e
100008	ca2faecdda2f22f988a078e30a85823e1c6d9b1f861bb76c6453f010c4ee702e	aa86b8adde58c28ecbb17df2cd4f149bbd15a3d5ca6ef7cc372ea180a5572246
100013	9f6d42070389b589f1a7569ce119bbcfbef2bf54986de4b6a392dd84a448f9bb	7f7aff125e964f5df0771c00a90ea972f367acbe5f16f02b67492e438550d33b
100016	defa79348d2290d9efc5eeccedeb31132e14ad00fe734110f535497c6d5493ca	46679c6cc522d854696d24d9eb77d75cdf60cc388ce825a451633cbfea960f77
100049	750fc7c7ad40737aec1e9852daa144f40a18a912b67d1754ef97a20adbe0b69c	ab83727df0045c5976dbbf3c1c330c7a9776f6120e0d48d5fefb5df94d59b708

9-19 of 45

Figure 5.1: User cast verification

Your shuffled ballot's encryption pair :

5954de2be853df67320621236b581bc3b4e2ade229976cc4c4f33927bf742989

97347646f61ee39315762b65a44b8a72ec23a8b6882cc9ca788188c7acbd1a86

All Fields Search

Alpha	Beta
101dd9e8fbb5ab7ea3f1136663557b07dcd9feef8b8d5fba49ab28dc3509cd48	346e12f5378a07fe1263089f100bd8c94ec9a02191f4e913e36504913411d7d9
e6860c23972191f02636f8cc714e8cd1e1020b30c27342c6ef0554ddd5825f06	cc626745223504d05acf03fb78c79e3fd93dd7914dfe9d97d741ed756fc32731
db03c1d238bf7b7a146a36ab4ddcc59e0c89e283056128940d2a309330665ee9	38e197a9934d1b9491fd0b06a64f8f69eea2bee683077f3efd2b1c48222dfc76
6b3b2069468855bf1fa8482c447dd9a788d8dff01fda8666e7894cb92e1293cf	614f2c539c62756049eb9180dc3785c8d64bdc0c0bfaa610bf0ef06e0fafdaad
5954de2be853df67320621236b581bc3b4e2ade229976cc4c4f33927bf742989	97347646f61ee39315762b65a44b8a72ec23a8b6882cc9ca788188c7acbd1a86
3e43a5e2a39e50a61c4836b7dc97473db4eab8c6ff90a23b33aa80c5ae858904	f310c81cdc1d0a6210bc2a4c978fab1adab93bf25b6ccdff1589768378a8433d
8cc186f786993a05e6dd6727ed0f75806ed81bfcdedf5cae3bec98b409fffa90	1bcbdddd5fa4c80b40203e3fcf1161287d6747c304822f61684f61a5ee3bd8c1
2b22d09971b90b2a0c8d84ba6a558c95ab34b439631946668fce53fff9e6cd01	231f92787cbccc44c87226fb01457dd1130d780ba5b6e7baae37d5fcac4fbc4c
a486e41e4070d2a67c47677940fe481f367f9be9be90dd8b96d260330220323e	2aad297b4de6c8dbb6de11827a903623520bf3aa4b07242e8407cca178cc2faa
6a443e8b07f3161a47518e41a476b82524f825b1cf72e43e8190625719bd362a	7dc95d62911d2a995417e9d54608ccefb3a9781e9a46bbb8d3ce94ffb5463424
613838367da55acae5ca9da56e6a53656a985eabf4bd6d009cad235dc8a2d391	d2325f96574370b63ee367c7212d7561d09f06f1857b1e4cb2843d22cfdbd20c

Record ID: 19 — 16-26 of 45

Figure 5.2: User shuffle verification

Your vote : 456789

All Fields Search

Place	Sciper	Votes
1	456789	10
2	123789	7
3	123456	6
4	248635	5

1-4 of 4

Figure 5.3: User result verification

6 Conclusion

The application presented is the basis of a project which will continue to grow during the next semester. The foundations lay on the three principles of internet security which are privacy, authenticity and integrity :

- **Privacy** is ensured by the various encryptions, first by the end-to-end encryptions on the communications links, secondly by the ElGamal encryptions of the ballots and thirdly by the Neff shuffle.
- **Authenticity** is brought by the authentication server and the Schnorr signature it produces.
- **Integrity** is ensured by the skipchain and the conodes making the whole storage reliable and immutable. Even if some servers are corrupted, thanks to the cothority principle, the result of the election will not change as long as at least half of the servers are reliable.

However the application needs some modifications before its release. As we saw earlier in *section 4* it already suffers of some limitations, in particular the Node.js authentication server does not suit the decentralization idea behind the cothority and would ideally need to turn into a distributed authentication.

Also, since theoretically ten thousand people should be able to use it at the same time, making sure that the application scales and is resilient to various types of attacks before its release becomes crucial.

By the end of the Spring, the EPFL should see coming this brand new evoting system, secured and scaled, specific to the school which will be able to rely on for its next elections.

Get the github's repository :

```
git clone https://github.com/dedis/student_17_evoting_frontend.
cd student_17_evoting_frontend/E_Voting_EPFL.
```

The setup of the authentication server and the frontend will assume that you are in the folder E_Voting_EPFL.

Authentication server

Be aware that the authentication server can only be launched when connected to the EPFL Wi-Fi or using the EPFL VPN.

```
cd authentication_server
node server.js
```

The console indicates : Server listening on port 3000 when the server is ready to be used.
To stop the server, CTRL-C.

Launch cothority and register skipchain ID

This part requires the project student_17_evoting from the dedis' Github repository developped in parallel to this project.

```
go get -u github.com/dedis/student_17_evoting
git clone https://github.com/dedis/student_17_evoting
cd student_17_evoting
go test -v ./..

### Creation of the master skipchain
''' cmd
./setup.sh run 5 3
cd cli
go run cli.go -roster=../public.toml
# Create the master Skipchain.
# Key has to be in base 64 representation.
# Admins has to be listed of comma-separated numbers, i.e 100, 200, 300
go run cli.go -pin=[pin] -roster=../public.toml -key=[frontend key] -admins=[list of
    admins]
# If the creation was successful the identifier of the master Skipchain is returned.
```

You have to recover the identifier if the master Skipchain and copy it in the config/config.ini file in the constant masterPin.

Frontend

The frontends are in the file client/html.
For the administrator frontend, launch web_admin.html with your favorite browser.
For the user frontend, launch web_user.html with your favorite browser.

[1] Helios Voting. https://heliosvoting.org, last viewed on December 31st, 2017.

[2] B. Adida. Helios: Web-based open-audit voting. In Proc. USENIX Security, Aug. 2008.

[3] C.P Schnorr, Efficient Identification and Signatures for Smart Cards. In "Advances In Cryptology - EUROCRYPT '89", LNCS 434, pp. 688-689, 1990.

[4] Taher ElGamal (1985), A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory. (conference version appeared in CRYPTO'84, pp. 10-18)

[5] C. A. Neff, P. Samarati, "A verifiable secret shuffle and its application to e-voting", 8th ACM Conference on Computer and Communications Security (CCS-8), pp. 116-125, November 2001.

[6] C. Andrew Neff. Verifiable Mixing (Shuffling) of ElGamal Pairs, April 2004. Available online on http://courses.csail.mit.edu/6.897/spring04/Neff-2004-04-21-ElGamalShuffles.pdf.

[7] B. Ford et al. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In 37th IEEE Symposium on Security and Privacy, 2016.

[8] B. Ford et al. CHAINAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. Proceedings of the 26th USENIX Conference on Security Symposium, June 2017.

[9] C. Lecommandeur, Tequila - A distributed Web authentication and access control tool. Available online on http://test-tequila.epfl.ch/eunis-2005-lecommandeur.pdf, last viewed on December 31st, 2017.

[10] Passportjs. http://www.passportjs.org/, last viewed on December 31st, 2017.

[11] Express. http://expressjs.com/, last viewed on December 31st, 2017.

[12] Passport-Tequila. https://github.com/epfl-sti/passport-tequila, last viewed on December 31st, 2017.

[13] LDAPjs. ldapjs.org, last viewed on December 31st, 2017.

[14] Protobufjs. https://github.com/dcodeIO/ProtoBuf.js/, last viewed on December 31st, 2017.

[15] W2UI Grid. http://w2ui.com/web/docs/1.5/grid, last viewed on December 31st, 2017.