# Mock Election

Andrei Florian

## Table of Contents



## Abstract

To test the functionality of the application, a mock election was conducted simulating how a real election would unfold on the application. This document will follow the process step by step, discussing the stages in the process.

## Mock Election Conduct


*Figure 1 - Mock Election Process*

Figure 1 illustrates the steps that will be conducted as part of the mock election.

### Create User Accounts

35 user accounts have been created using the application. To speed the process up, these accounts were created directly in MongoDB by adding records to the authentications collection.

One of the 35 accounts is an admin account that is used to manage the election. The other 34 accounts are user accounts. 5 of the 34 user accounts are not verified (the voter did not verify their ID). The election is to take place over 3 constituencies: Dublin Central, Dublin East, and Dublin West.

Each constituency has 10 of the 29 validated user accounts allocated to it (the last having 9). The authentications record is available to download as a separate file in this directory.

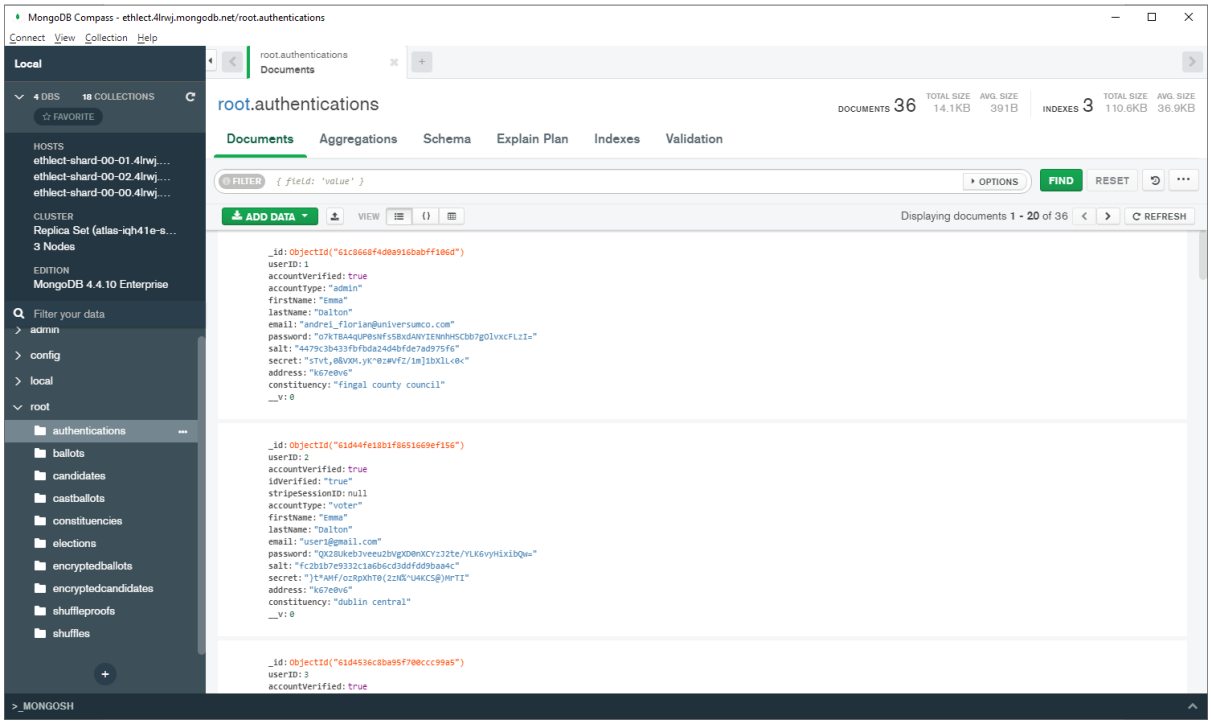


*Figure 2 - MongoDB Authentications Record*

## Create Election

I logged into the application using the admin account and navigated to the admin page. From there, I entered the "Create New Election" page and entered the following details on the details tab:

| Field | Data |
|---|---|
| **Election Name** | Mock Election |
| **Election Description** | A mock election. |
| **Election Start Date** | 10/01/2022 |
| **Election End Date** | 17/01/2022 |
| **Constituencies** | Constituencies Seed File (available in this directory as a csv file) |

*Table 1 - Election Details*

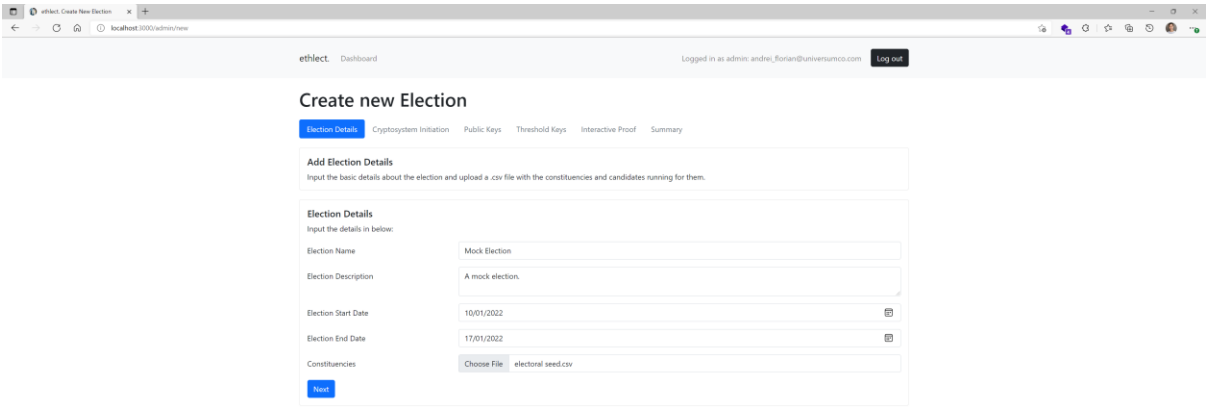


*Figure 3 - Election Details Page*

I set the generation mode to remote on the next tab. I set the threshold value for the threshold keys to 2 to make testing faster (reduces the number of manual cryptographic functions I need to perform on the keys). I used the keyholders app (available in the root directory of the repo) to generate one RSA keypair, the keypair is also available as a file in this directory.

I will use a single keypair for all keys. This is so that I don't need to manage a different keypair for each key inputted.

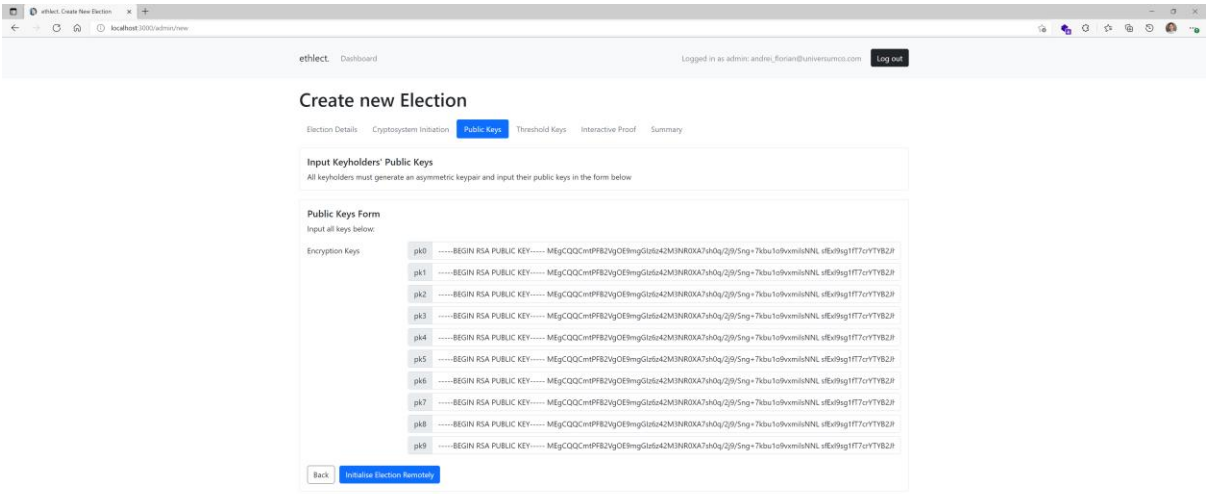


*Figure 4 - Public Keys inputted*

The application generated the election in 9 seconds since the submittal of the keys and outputted 10 threshold keys encrypted with the public key provided.

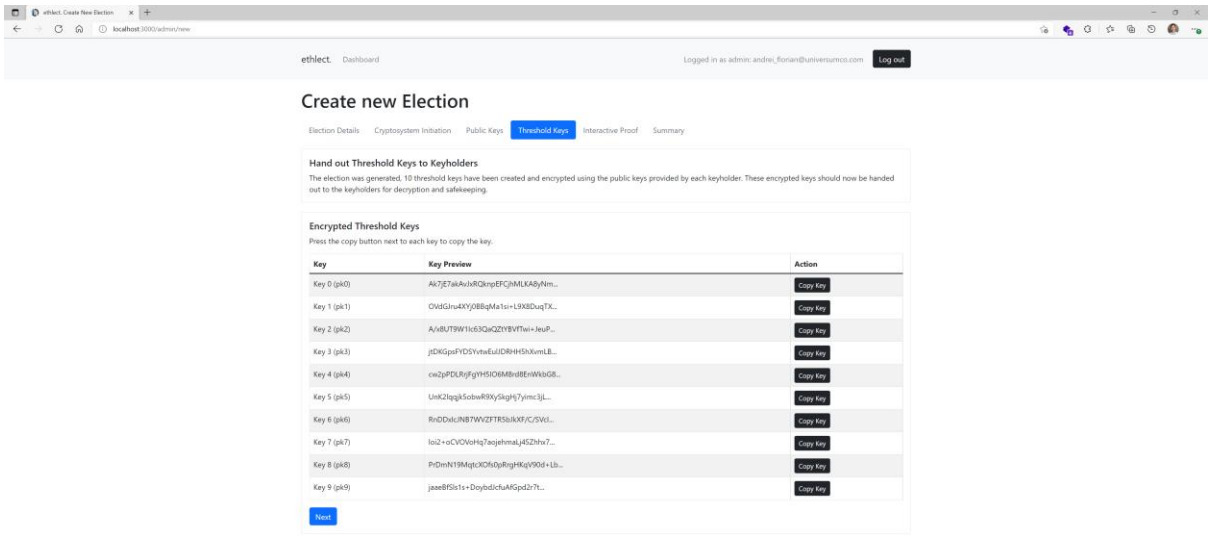


*Figure 5 - Encrypted Threshold keys*

Because I am working with 2 keys, I only copied the first two keys. I then used the keyholders application to decrypt these keys using the private RSA key. The resulting decrypted keys are in this directory.

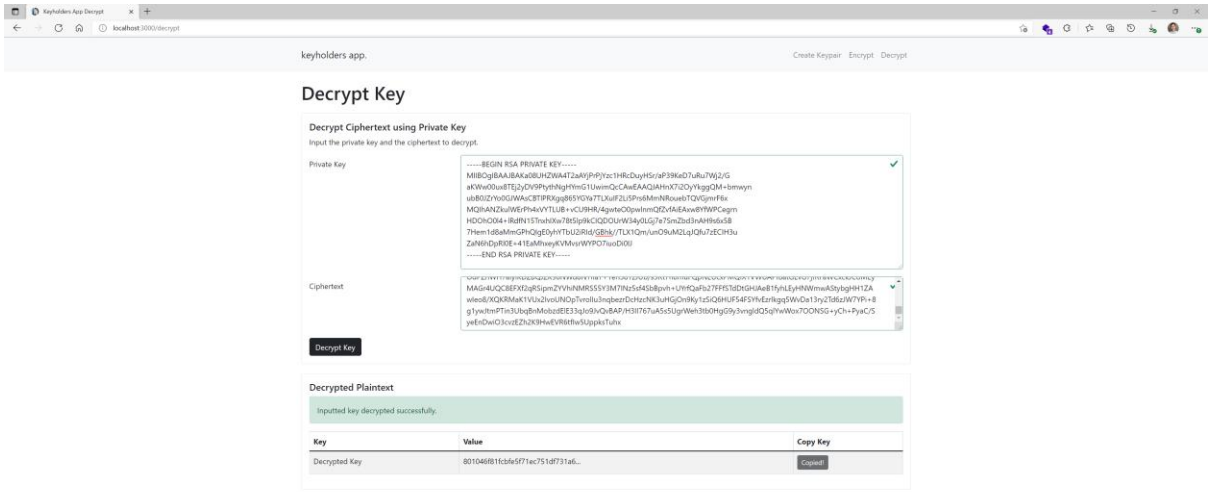


*Figure 6 - Decryption using Keyholders App*

I then encrypted the 2 keys with the election public RSA key exposed on the next tab of the form (the encrypted keys are also available in a file in this paper's directory). I copied the first key in the first field and the second in the remaining in the interactive proof tab.

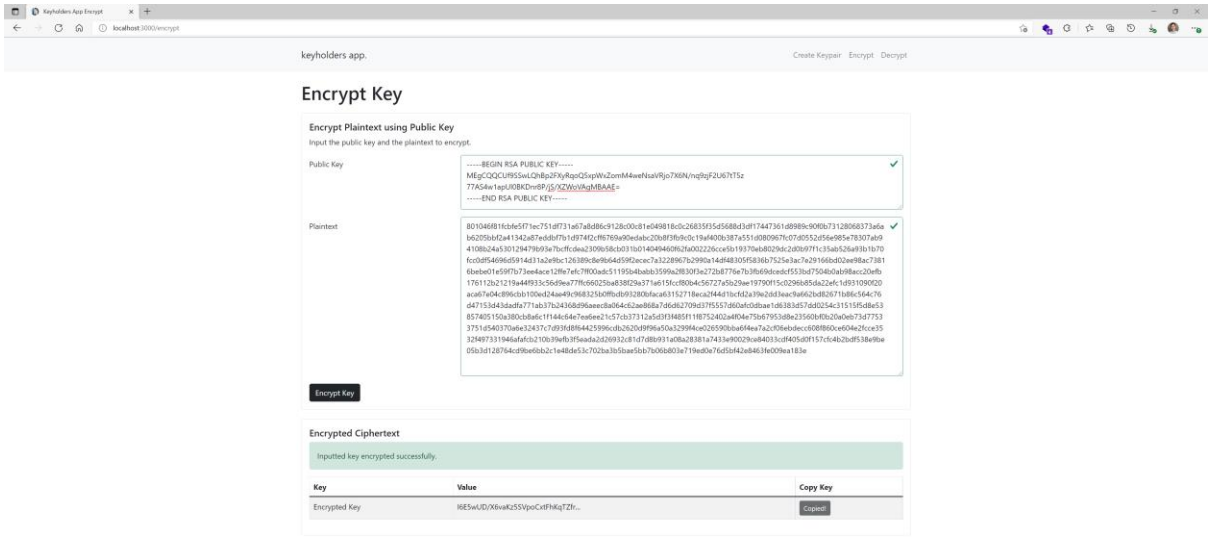


*Figure 7 - Encryption Using Keyholders App*

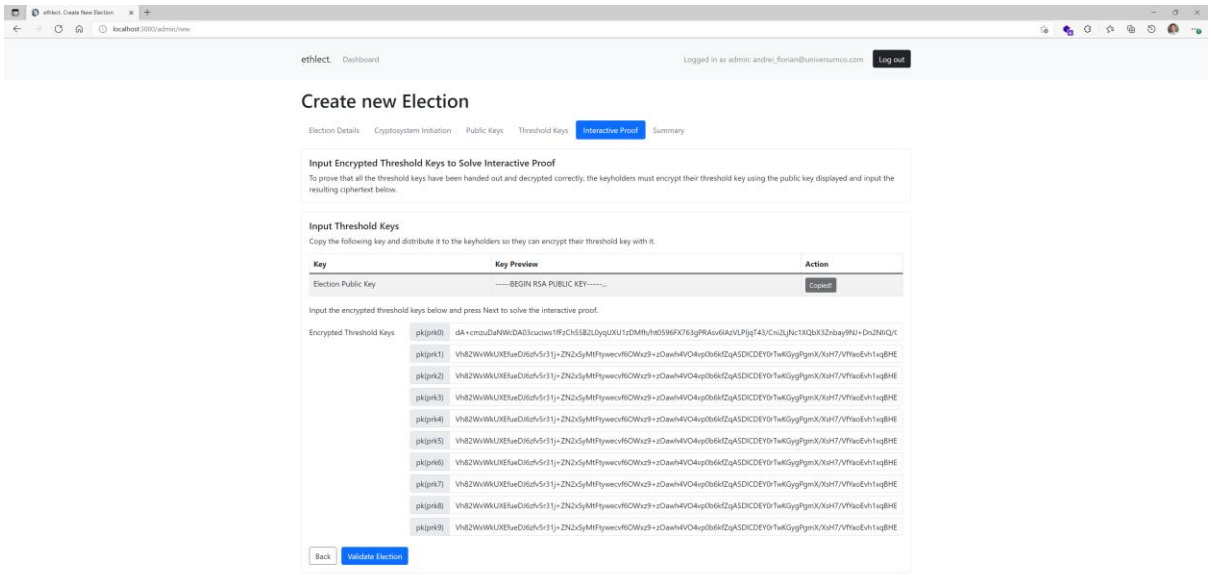


*Figure 8 - Keys Proof*

Finally, I submitted the form, and the election was successfully validated.

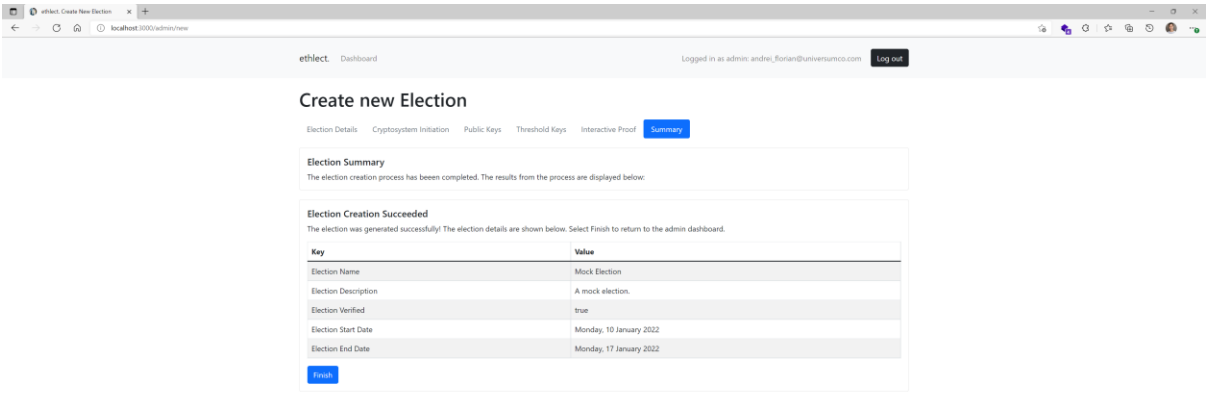


*Figure 9 - Election Creation Summary*

I then generated ballots for the election by heading to `/admin/${electionID}/generate` and pressed the generate button.

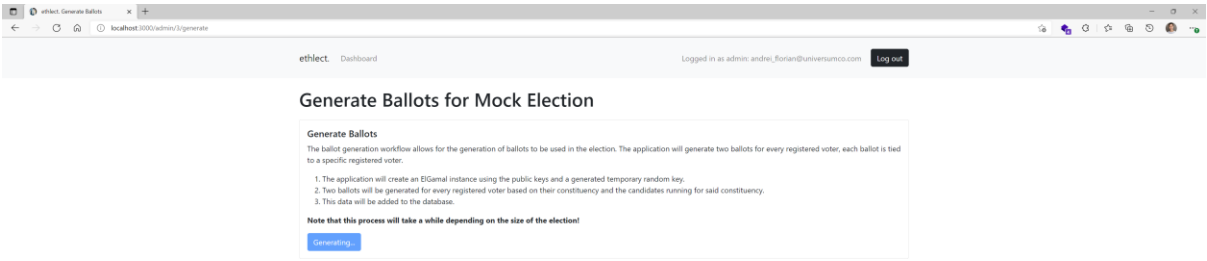


*Figure 10 - Ballot Generation*

The process took 54:24 minutes and the application successfully generated 58 ballots (2 for every verified voter).

## Decrypt Ballots

I simulated the decryption and printing process by manually decrypting the ballots in order to reveal the relationship between the candidate representative number and

candidate. These decryptions were done manually by inputting the encrypted candidate IDs into a decryption function (akin to the one used in the app's decryption process).

After decrypting all the candidates, I added the candidate reps of the candidates paired with the candidate IDs they represent for one of the ballots assigned to each voter to an excel spreadsheet. I only added 2 such pairs at random per ballot to make casting the ballots faster (since it would take a long time to input more candidates when casting a vote). This spreadsheet is also made available.

## Cast Ballots

After the ballots were decrypted, I started the electoral period to start accepting votes for the election.

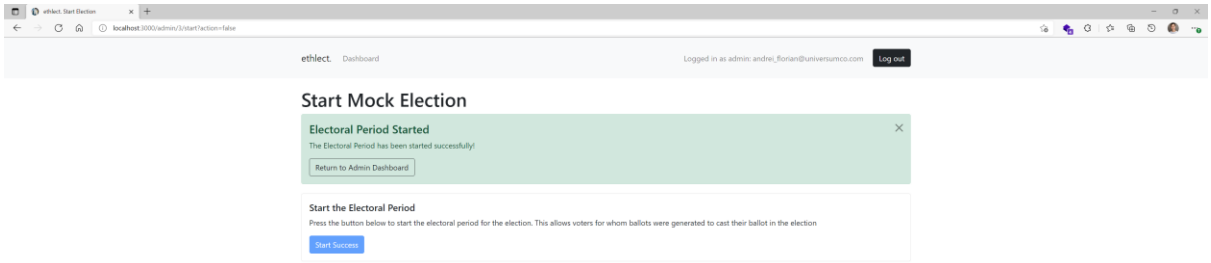


*Figure 11 - Electoral Period Start*

I logged in, with the help of my family, into every one of the validated accounts and cast a ballot consisting of the candidate reps in the ballots cast excel spreadsheet. Overall, all 29 registered voter accounts cast a vote.

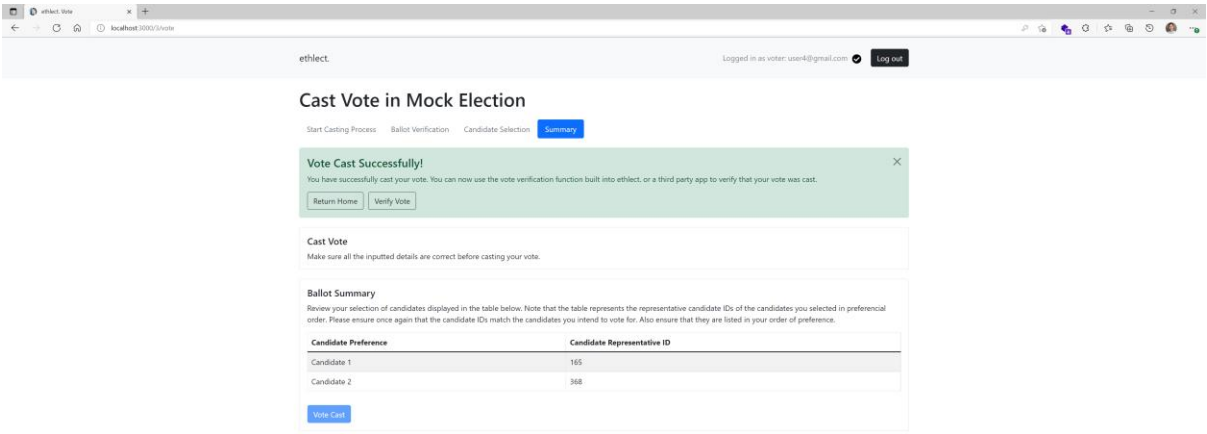


*Figure 12 - Vote Cast Successfully*

I also tried casting a vote with the accounts that were not verified and received an error. I also tried inputting the wrong ballot ID, invalid candidate IDs, duplicate candidate IDs and no data, which all resulted in errors.

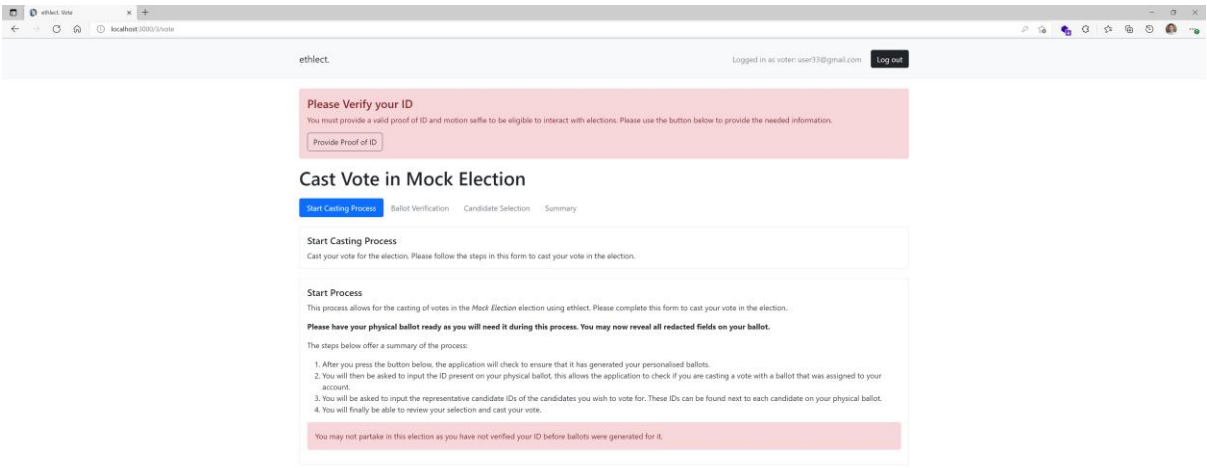


*Figure 13 - Error Casting Ballot*

All voters also received positive confirmation that their ballot was recorded correctly by verifying their ballot directly by visiting the page `/${electionID}/verify`.

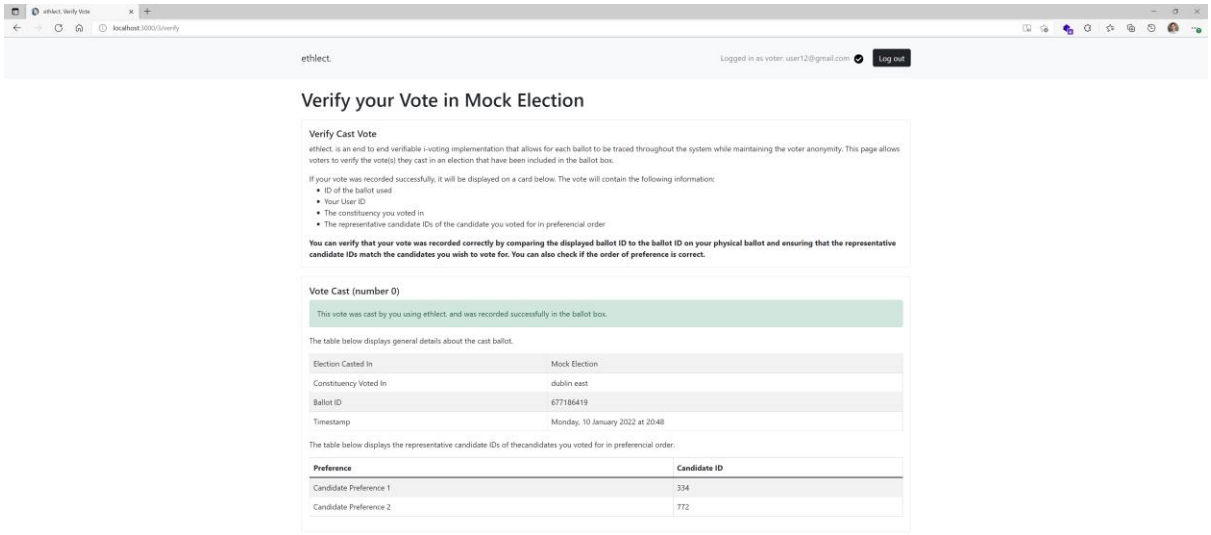


*Figure 14 - Voter Verifies Vote*

## Shuffle Ballots

After all ballots were successfully cast, I logged back into the admin account and started the tabulation process by heading to `/admin/${electionID}/tabulate`. This ended the electoral period (voters could no longer cast votes) and allowed me to create shuffles.

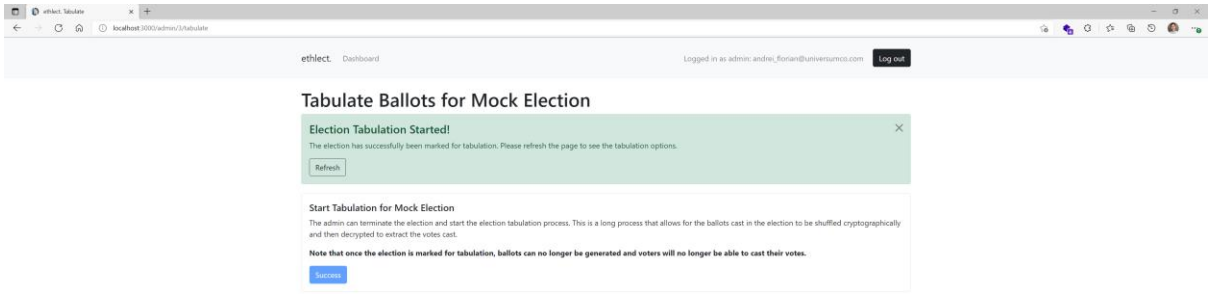


*Figure 15 - Start Tabulation*

I created a new shuffle by navigating to `/admin/${electionID}/tabulate/shuffle` and pressing the “Shuffle” button. The ballots were shuffled successfully, and the process took 11:54 minutes to complete.

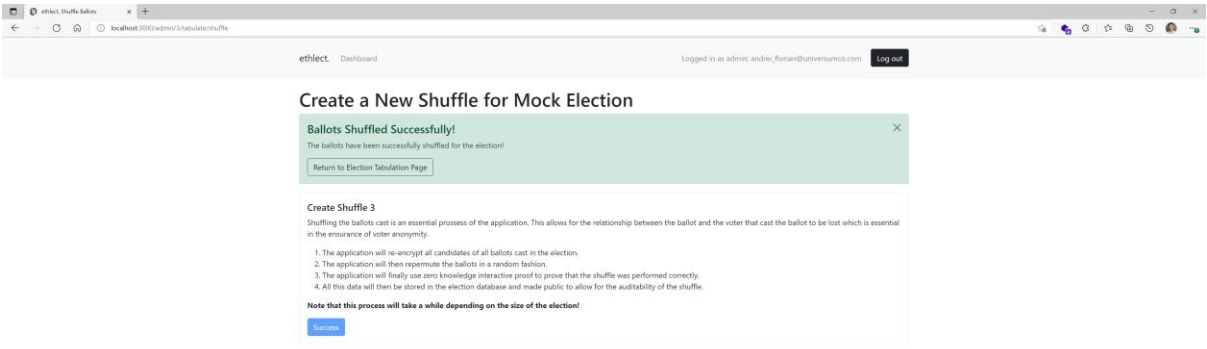


*Figure 16 - Successful Shuffle*

I then returned to the tabulation page and approved the shuffle to be able to decrypt the ballots.

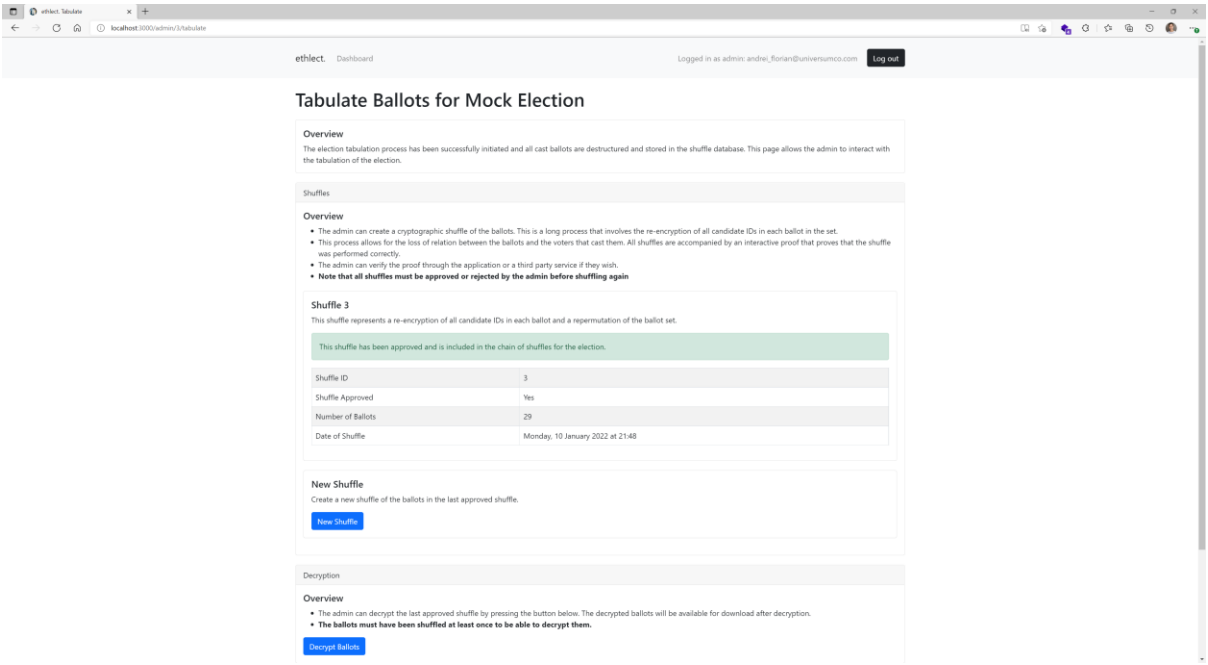


*Figure 17 - Approve Shuffle*

## Decrypt Ballots

After the shuffle, I headed over to `/admin/${electionID}/tabulate/decrypt`. I encrypted the threshold keys for both keyholders using the provided election public key using the keyholders app again and inputted the keys into the form. I then pressed “Decrypt” to decrypt the ballot set.

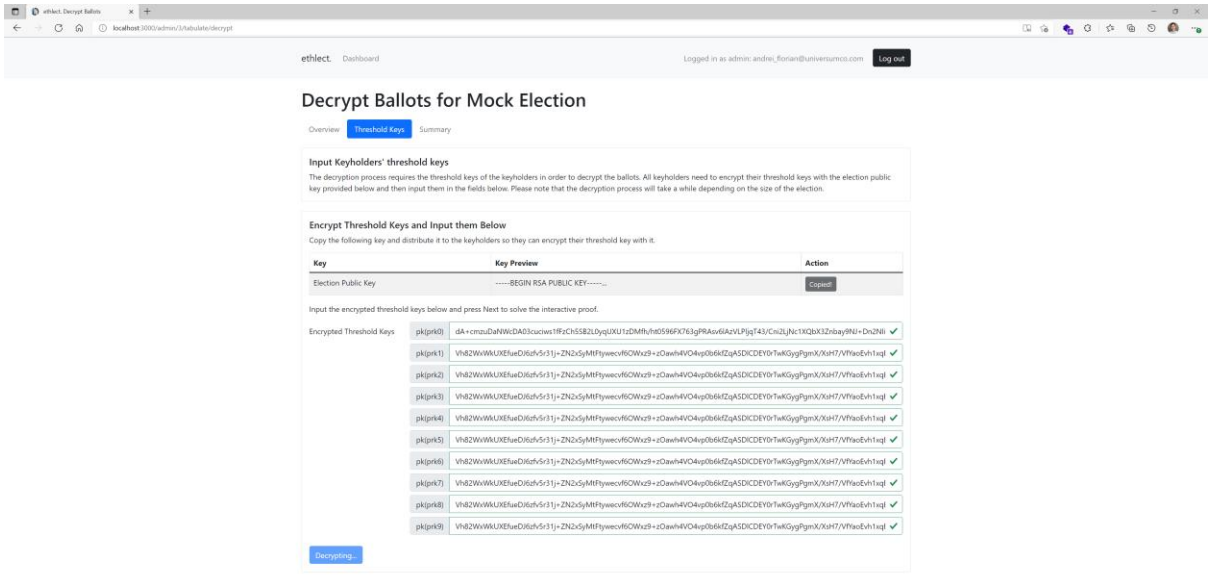


*Figure 18 - Start Decryption Process*

The ballot decryption process succeeded in 4 minutes and 10 seconds. This finalised the election. I finally checked the results outputted by the decryption against the excel spreadsheet created to ensure that the decryptions match. All candidates matched which in turn confirmed that the ballots were correctly processed throughout the system.

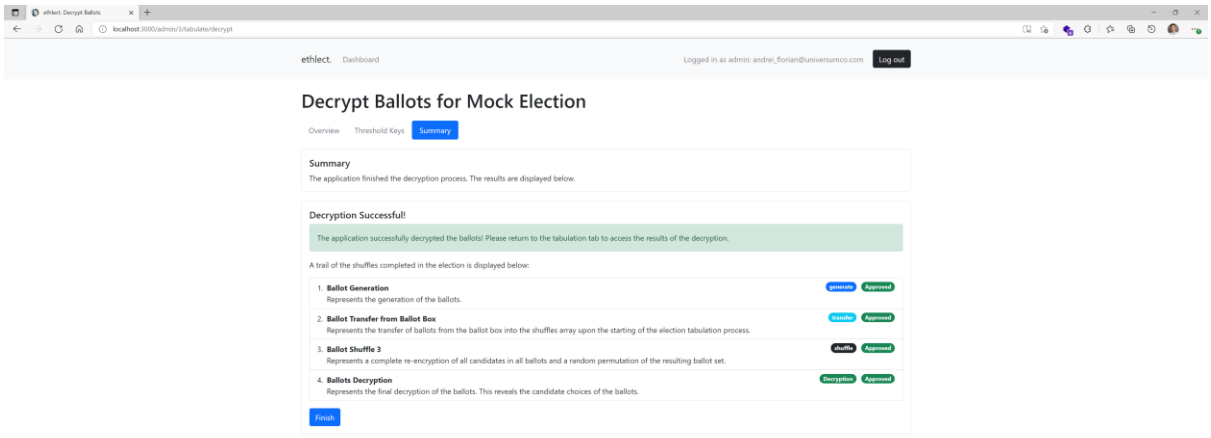


*Figure 19 - Ballots Decrypted Successfully*

The results of the election and the entire election document from the database can be accessed from the directory of this paper on GitHub.

## Prove Proofs

After the election was completed and the results were proven to be correct, I went to the election's audit page at `/${electionID}/audit`. From there, I downloaded all the proofs generated by the application:

| Proof | Description |
|---|---|
| **Ballot Generation** | The proof that proves the correctness of the ballot generation |
| **Ballot Transfer from Ballot Box** | The ballots transferred from the ballot box into the shuffles array |
| **Ballot Shuffle** | The proof that proves the correctness of the shuffle |
| **Ballot Decryption** | The proof that proves the correctness of the decryption |

*Table 2 - Proofs outputted by the application*

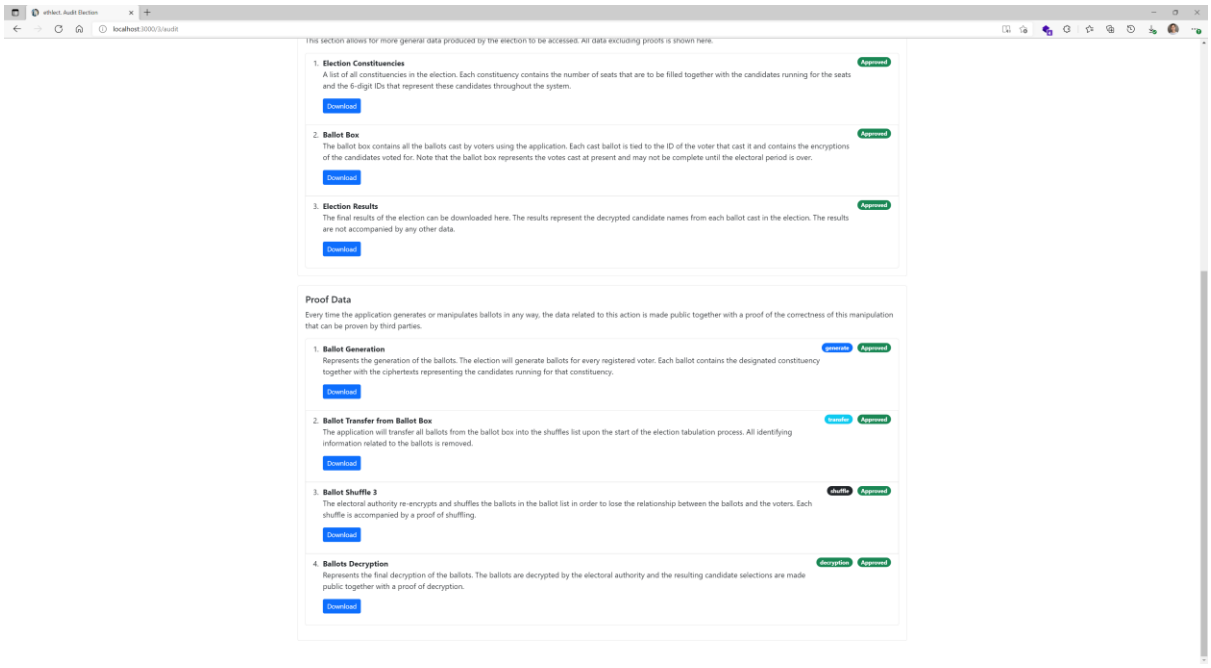


*Figure 20 - Election Audit Page*

All these proofs are available in this directory. I then inputted the ballot generation, shuffle, and decryption proofs into the audit app by uploading the downloaded files in the form in the app. I checked the transfer proof manually by ensuring that all candidates in the output of the proof are present in the ballot box and are in the right order. This checked out.

| Proof | Duration (minutes) |
|---|---|
| **Ballot Generation** | 39:24 |
| **Ballot Shuffle** | 11:23 |
| **Ballot Decryption** | 2:35 |

*Figure 21 - Duration and results of proofs*

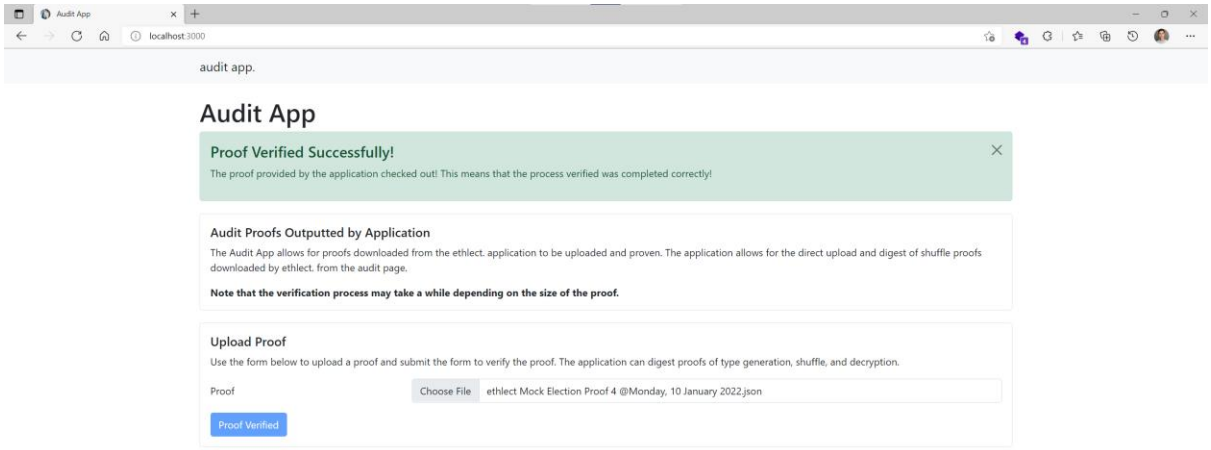


*Figure 22 - Proof Proved Successfully*

# Conclusion

In conclusion, the application was proven to work perfectly. While testing the application, I encountered no unexpected errors. For context, the application was run on a remote computer hosted by Microsoft through Windows 365 running with 4 CPU cores, 16GB RAM, 128GB storage (note that the application was run alongside other programs). The mock election conduct is summarised below:

1. I created 35 user accounts. One of these accounts was an admin account, and the others were voter accounts. 29 of the voter accounts had their ID verified and the remaining did not to allow me to test to ensure that a voter who has not proven their ID cannot vote.
2. I created a new election on the platform, generated 10 threshold keys and set a threshold value of 2. I used the keyholders app to generate an RSA keypair, decrypt the threshold key received from the app, and re-encrypt it using the election's public key and pass it to the app again. I then generated the ballots and ensured that they were only generated for verified users.
3. I decrypted the ballots manually and paired the representative candidate IDs with the decrypted candidate IDs for all candidates on all ballots. This is a simulation of the ballot printing process and allowed me to know which candidate each candidate rep pointed to so I could ensure that the correct candidates were being outputted after the decryption.
4. I then signed in with all the voter accounts created and cast the ballot assigned to them based on the excel spreadsheet I created during the decryption process. I

tried inputting incorrect data into this process to ensure that the application flags the error.

5. I started the tabulation process through the application and shuffled the ballots once. I approved the shuffle thereafter.
6. I finally decrypted the ballots through the application and checked that the outputted candidate IDs matched the selections inputted during the casting process by cross-referencing them to the excel spreadsheet I created then.
7. I finished the mock election by verifying all proofs generated either manually or by feeding them to the audit app created. All proofs checked out proving the correctness of all processes in the election.

All documents generated during the mock election are available in the resources directory alongside this paper. These documents allow for the independent verification of correctness of the mock election.