Bachelor Thesis

Polar - a Centralized Digital Participation Platform for Political Groups and Organizations.

Simon Bärmann

Gutachter: Prof. Dr. Hannah Bast
Betreuer: Patrick Brosi

Albert-Ludwigs-Universität Freiburg

Technische Fakultät

Institut für Informatik

Lehrstuhl für Algorithmen und Datenstrukturen

07. Februar 2024

Bearbeitungszeit

09. 11. 2023 – 09. 02. 2024

Gutachter

Prof. Dr. Hannah Bast

Betreuer

Patrick Brosi

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my thesis has not been prepared for another examination or assignment, either in its entirety or excerpts thereof.

Freiburg im Breisgau, 07.02.2024

Place, Date

[illegible]

Signature

Abstract

Political groups and organizations, in particular small ones, struggle with using technology for their political practice. Incomplete email lists or Facebook groups often define which member has access to group-related information. Internal elections are still frequently conducted in person or with insecure tools only. We present Polar, a centralized solution for digital participation, and Eos, a transparent and secure election service. Polar provides functionality for the typical workflows of political groups, such as secure chat rooms (through Matrix integration), motion creation and discussions and an event system. It is integrated with Eos to perform secure elections, making use of Polar's single-sign-on functionality. To validate the conception of both applications, we evaluate Polar and Eos through a user study, which resulted in predominantly positive feedback.

Zusammenfassung

Politische Gruppen und Organisationen scheitern häufig daran, ihre Arbeit effizient und effektiv ins Digitale zu verschieben. Häufig verwenden sie datenschutzrechtlich-fragwürdige Anbieter, um die Kommunikation mit einem (unvollständigen) Teil der Mitglieder zu ermöglichen. Wahlen werden entweder gar nicht oder mit unzureichend sicheren Tools im Internet durchgeführt. Als Antwort auf dieses Problem stelle ich in dieser Arbeit eine Softwarelösung bestehend aus zwei Programmen vor, die sowohl eine digitale Partizipationslösung als auch eine sichere Platform zum Wählen bieten: Erstens, die Platform Polar ('Politische Arbeit'), die es politischen Gruppen ermöglicht, gemeinsam Anträge zu sammeln und zu bearbeiten, Events zu teilen, miteinander zu kommunizieren und sich bei anderen Diensten mit ihrem Polar-Account anzumelden. Zweitens, Eos ('Election Online Service'), mit dem sichere und verifizierbare Wahlen durchführbar sind. Dabei werden die Stimmen mit dem Eos Ver- und Entschlüsselungsprotokoll verschlüsselt, welches so konzipiert wurde, dass es von möglichst vielen Menschen verstanden werden kann. Eos unterstützt Drittpartei-Trustees, die sicherstellen, dass selbst korrupte Teilnehmer am Entschlüsselungsverfahren die Integrität nicht gefährden können. Mittels einer Nutzerstudie wurden die beiden Tools evaluiert. Diese Studie kam zu dem Ergebnis, dass beide Tools für Engagierte im politischen Kontext sehr relevant sind und sich die Studienteilnehmenden vorstellen konnten, die Software zu verwenden.

Contents

1 Introduction

1.1 Motivation

When the Covid pandemic brought the world to a standstill in 2020, political organizations and groups were strongly affected and suddenly had to move their work online. These groups had to find ways to communicate with each other. They often turned to video chat services like Zoom or Jitsi, messaging services like Telegram or Signal, or Email distribution lists to stay in contact. Still, many of these groups were often overwhelmed by this, as can be seen in the example that political groups were very soon exempted from the contact restrictions and curfews for their work, at least in Germany. Digital solutions for essential tasks of political organizations and groups should and neither can replace personal collaboration, but well-thought-out approaches can facilitate digital and offline work not only in times of a pandemic.

1.2 Objective

The objective of this thesis is to present a software solution that attempts to solve this problem. This software should offer suitable functionality for the essential tasks of political groups and organizations. These include the procession of motions (to create resolutions, compile electoral programs), intra-organizational communication among members, the collection and sharing of events, and the conduction of (secure and verifiable) elections. This software solution then is to be evaluated by externals in the form of a user study in order to understand whether the software solution meets the requirements of political groups and whether the apps appeal to users.

1.3 Approach

In this thesis, we present two applications designed to simplify and centralize the internal work of political groups in the digital space. Eos ('Election Online Service') is an app to conduct encrypted and verifiable elections. Polar ('Politische Arbeit') is an app for political organizations to centralize and digitalize their workflow. Polar provides the functionality for collaborative collection and processing of motions, for collection and sharing of events and for chatting with other members. It can also be used as a Single-Sign-On provider, which means that users can use their Polar account to login at other services (like Eos or NextCloud).

Eos and Polar use the same underlying frameworks and languages. Both are separated in server (backend) and client (frontend) software. The backend is written in Python 3.11 and the frontend is written in Dart. Both backends use FastAPI [1], a "modern, fast (high-performance), web framework for building APIs". Both use SQLalchemy [2] as their object-relation mappers (orm) framework, allowing a 'high-level abstraction' [3] of and access to the data in the relational databases. Both FastAPI and SQLalchemy are commonly used and well-tested frameworks, making them a reliable choice as core frameworks. The frontend uses Flutter [4], a relatively new framework which allows to develop for mobile, web and desktop environments on a single code base.
The idea behind the conception of Eos and Polar as two separate applications is that Eos could be used outside the scope of political work. As a standalone application it is thus available to more users.
We evaluate the applications by conducting a user study in which the participants were allowed to test the apps live while the conductor guided them through the

different features and asked them survey questions in the process. In this study the apps were rated highly for their design and user experience. The vast majority of study participants trusted Eos. Polar's features were consistently described as useful and necessary. All politically active participants could imagine using Polar in their organizations.

1.4 Structure of this Work

In Chapter 2 we will take a look at related work on both the question of secure elections as well as software solutions for digital political participation. Then, in Chapter 3, we will look at an overview of the features and concepts driving Eos and Polar. In the following theoretical analysis (Chapter 4), we will pay a closer detail to algorithms like Polar's close event algorithm, the Eos' en- and decryption protocol and others.

Following this chapter, in Chapter 5, the apps will be evaluated using the conducted user study. Then we will take a look at the limitations of both apps, and analyze ways Eos could be attacked. See Chapter 6 for this. Further, we will comment on future work that needs to be done to enrich both apps. Finally, in Chapter 7, we will conclude in final remarks on both Eos and Polar and decide whether or not the objective of this work was met.

2 Related Work

In this chapter, we will take a closer look on existing concepts and software solutions for (secure) digital voting systems and for digital participation.

2.1 Digital Voting Systems

Helios Voting is a paper and a software solution focused on 'open-audit voting' [5]. Open-audit means that it enables everyone to verify the election. It introduces the following concepts to make online elections secure and verifiable: Helios allows everyone to fill out a ballot, encrypt it and analyze this encryption. Authentication is only required for the actual casting of votes. For the verification of the own encrypted ballot, all necessary secrets are displayed to the user so that these can be copied into third-party tools to ensure the correctness of the encryption process. The proposed encryption protocol was based on shuffle-encryption [5], but later was changed to an homomorphic encryption protocol [6]. Homomorphic encryption protocols have the advantage that a single ballot never has to be decrypted as homomorphic encrypted ballots (if encrypted correctly) can simply be added together arithmetically in their encrypted state and then be decrypted correctly as one. Helios keeps track of who the owner of an encrypted ballot is to make sure that no one votes twice and releases a 'fingerprint' with which the user can track their vote in the tally [5]. Helios is still available on the internet (`https://vote.heliosvoting.org`) and over two million votes, from around the world, were cast using their system. In their paper the authors argue that only one trustee (the Helios server itself) is necessary to ensure the integrity of the election. They argue that even a fully corrupt Helios server could not fool everyone to believe that the results of a faked tally are actually correct. Because a tally is fully auditable, corrupt administrators could not fake the election results without experts being able to detect their malicious access [5]. Today, Helios supports multiple trustees ensuring the integrity of an election. Homomorphic encryptions can be used to decrypt the sum of all encrypted ballots, but homomorphic encryptions still allow for any vote to be decrypted directly. If the helios server has access to all secrets needed, for example when no trustee is present or if all trustees are corrupt, the vote of a voter still can be decrypted directly.

Next to Helios there are multiple commercial online election services available. These are usually only available to users after a one-off purchase or subscription, connecting their prices to the expected amount of voters for elections. The main target group for such services are companies. Tools like nemovote [7] and electionbuddy [8] or electionrunner [9] offer such services but neither one of them releases how they exactly encrypt their cast votes. This is as if voters would throw their ballot into a hole in the wall, not being able to see what the people behind the wall are doing to the ballot. electionrunner and nemovote also offer features for admins to see the results of an running election live. electionrunner states that they only update this list every one to five minutes to make sure that no one can see that 'Voter A has voted and then sees that Candidate C's vote count increased by 1'. This means that the secret of the election on electionrunner actively depends on at least two votes being cast within periods randomly selected by the server, whereby both votes must not be identical so that it is not possible to determine beyond doubt what the voter voted for.

There are other tools like STARvote [10] which focuses on secure electronic voting systems. These are used for in-person voting. Electronic voting systems usually consist of dedicated machines only on which the voting procedure can be performed. Electronic voting is used in multiple countries around the world even for (very) high-stake elections like in the United States for the presidential election. Particularly

during the last U.S. presidential election, the use of voting machines fell into disrepute, especially as they were stigmatized as insecure by those who did not want to accept their defeat [11].
Digital voting systems exist for both in-person elections as well as for online elections. These online election services feature different designs, where some are very transparent and some not so much on how they encrypt the votes of the users. Helios has introduced many important concepts for the verifiability of online elections, but the encryption process is not necessarily understandable for the non-expert. Without the decryption of each vote, users must rely on mathematical proofs like zero-knowledge-proofs, with which the voters could be convinced that their vote was respected in the tally. It is open for discussion whether zero-knowledge-proofs really convince the non-experts on this matter.

2.2 Digital Participation Tools

Digital participation tools started with basic online forums. Nowadays they are used in municipalities, political parties and groups. One of the famous tools is LiquidFeedback [12] which was used in the German Pirate Party. This tool belongs to the group of "democrative delegative tools" because it does not only feature a grassroots approach to organize the party, but also allows members to delegate their vote to someone else. LiquidFeedback prominently features the ability of collecting motions in a four-step process lasting several weeks. Everyone can participate and vote on the adoption of the motion, but if wanted, they can also delegate their vote to someone they think would make a more profound decision. This could be members of a group inside the party focusing on the topic the motion is discussing about. A delegated vote increases the weight of the 'experts' vote [12]. LiquidFeedback's voting system does not encrypt votes. It saw a big rise in users during the 'highs' of the Germans Pirate Party, with 10.000 active users in months prior and following

important elections and successes. But with the decline of the Pirate Party also the interest in this tool vanished. The federal organization of the party does not use LiquidFeedback anymore.
The open-source software adhocracy [13] by the German 'Liquid Democracy e.V.' (not affiliated with LiquidFeedback) is used by the Berlin city government as an offer for public participation. 'mein.berlin.de' is a fork of adhocracy used to share information by the Berlin administration and for citizens to contribute their own ideas and suggestions for the city. The tool collects ideas for projects created by the administration all around Berlin. The citizens can suggest and discuss ideas, with over 76.000 ideas and comments already shared. The ideas are used to fuel decision-making in the administrative and in the legislative bodies of the city. This is also a form of grassroots participation platform [14].
Almost all big democratic parties in Germany use OpenSlides [15], an open-source tool to support in-person party conventions or assemblies by allowing hybrid participation. It is a web application to manage assemblies by providing features for managing the agenda, motions and elections of an assembly. Delegates do not have to be personally present in the assembly to participate in elections or submit amendments during the convention. Non-encrypted anonymous and personalized elections can be held digitally, allowing immediate tallying and live results for elections on motions and personnel during the convention. The feature to assemble a list of speakers during the ongoing assembly allows for a regulated discourse. Motions can be collected prior to the assembly and an agenda can be compiled that incorporates the motions received. OpenSlides is designed to be used per assembly. There is no real contingency between assemblies. Once an assembly is over, OpenSlides hibernates till the next assembly is on the horizon.
We can see that there is a variety of digital participation tools. They have multiple things in common. First, if they allow to vote on something, these votes are not encrypted. Even if they claim anonymous votes, a simple lookup in the database would revoke all secrecy. Second, the apps usually focus on one specific use case. But

the work of political groups does not end with assemblies and is not solely focused on motions. A digital participation platform that unifies the day-to-day tasks of political groups is therefore still missing, while existing solutions provide good inspiration on what such a tool should support.

3 The Apps

3.1 Polar

Figure 1: The logo of Polar representing the four features (motions, events, chats, SSO).

To summarize the concept of Polar, imagine it as a permanent online party convention. Party conventions are usually limited in time. They are representative through delegates and focus on a specific topic (for example, the determination of the election program for an upcoming election or the nomination of candidates). They usually consist, when looking at them in a procedural manner, of debates and motions. Motions are usually linked to the topic of the convention, and are debated on. Motions can receive amendments and on these amendments and the general adoption of the motion will be voted on by the delegates of the party convention. But party conferences also consist of informal discussions between delegates, which are not part of the formal organization of the convention.

Many internal organizational processes resemble this pattern, regardless of whether the political organization is a large national party or a small local association of a club of bicycle advocates, that just wants safer cycle lanes. They all have topics on which they have to make resolutions on. But to translate this workflow by copying it into the digital space would render invisible the advantages a digital system could offer. Rather than forcing members to sit in front of their device for a whole weekend, we propose to disintegrate the process in longer phases, a permanent online party convention enabling all members to participate. This idea is influenced by LiquidFeedback [12]. A disintegrated process allows members of Polar to engage whenever they have the time, be it on their way to work or in the evening at home. Polar summarized as a question could be understood as: 'What if we turn party conventions into a social media-like app?'.

3.1.1 The Motion Process

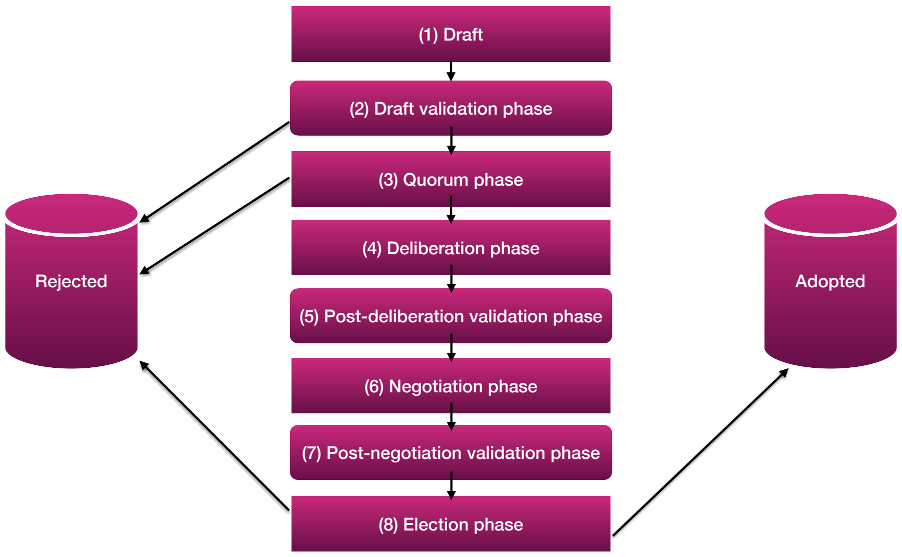

(1) Draft
(2) Draft validation phase
(3) Quorum phase
(4) Deliberation phase
(5) Post-deliberation validation phase
(6) Negotiation phase
(7) Post-negotiation validation phase
(8) Election phase
Rejected
Adopted

Figure 2: Illustration of the different phases of a motion. The order is not finally given, admins can set the phase of the motion freely, even if it already passed this phase before.

Polar allows to group motions in MotionGroups, they can be stacked and reach unlimited depth. Each MotionGroup has a MotionRules object, which defines which phases a Motion (in this group) usually goes through, how long these phases are and which obstacles a motion needs to pass. MotionGroups can share one MotionRule, meaning that for various groups one ruleset is valid.

The aim of a motion for a political organisation or group is to accumulate in a resolution, be it an electoral program of a political party or a resolution of the local student council. In the following, the process shall be explained using an imaginary context of a student council using Polar to allow all students enrolled in the university to participate in the democratic process.

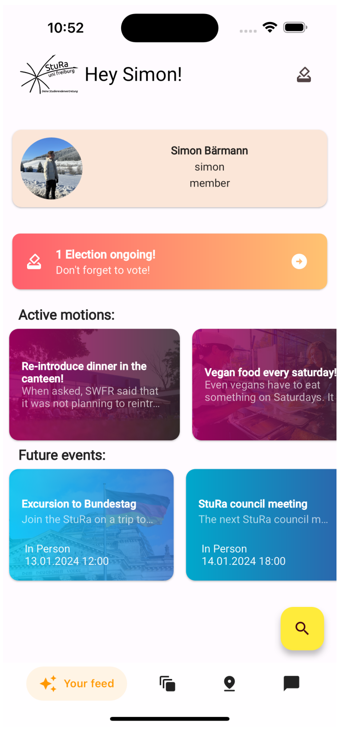
If a student now proposes a motion which calls for the introduction of an evening canteen, this motion runs through different phases (as shown in Figure 2 and illustrated using screenshots in Figure 3). The motion process uses ideas also used in LiquidFeedback [12].

1. **Draft phase**: Motions in this phase can still be edited by the student. The student sets a proposal and a description/explanation why they propose this motion. Once the student is finished with this, he can submit the motion. Motions in this phase are not featured to other users, but they could access them if they have the unique id of the motion.

2. **Draft validation phase**: The motion council (Polar users with the special right to edit motions) can decide on whether to allow this motion or not.

3. **Quorum phase**: If the MotionRule allows this phase, the motion now is looking for endorsers. Members can see the motion and endorse it, if they like it or want it to be dealt with. The MotionRule defines how many endorsers a motion needs to pass the quorum. In our context this means, if other students find this motion important, they can endorse it.

4. **Deliberation phase**: Once a motion passed the quorum phase it reaches the discussion phase, where members can discuss on the motion by creating comments. These comments contain a mood (from -2 to $+2$, where 0 means neutral), allowing them to display how supportive they are with the motion.


10:52

Hey Simon!

Simon Bärmann
simon
member

1 Election ongoing!
Don't forget to vote!

Active motions:

Re-introduce dinner in the canteen!
When asked, SWFR said that it was not planning to reintr...

Vegan food every saturday!
Even vegans have to eat something on Saturdays. It

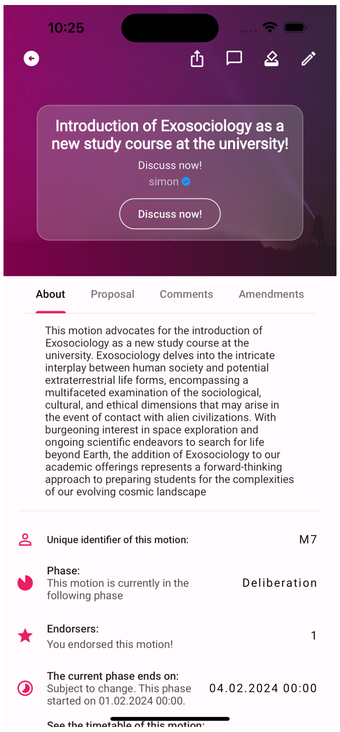
Future events:

Excursion to Bundestag
Join the StuRa on a trip to...
In Person
13.01.2024 12:00

StuRa council meeting
The next StuRa council m...
In Person
14.01.2024 18:00

Your feed

(a) The Polar main screen themed for the StuRa.


10:25

Introduction of Exosociology as a new study course at the university!
Discuss now!
simon
Discuss now!

About Proposal Comments Amendments

This motion advocates for the introduction of Exosociology as a new study course at the university. Exosociology delves into the intricate interplay between human society and potential extraterrestrial life forms, encompassing a multifaceted examination of the sociological, cultural, and ethical dimensions that may arise in the event of contact with alien civilizations. With burgeoning interest in space exploration and ongoing scientific endeavors to search for life beyond Earth, the addition of Exosociology to our academic offerings represents a forward-thinking approach to preparing students for the complexities of our evolving cosmic landscape

Unique identifier of this motion: M7

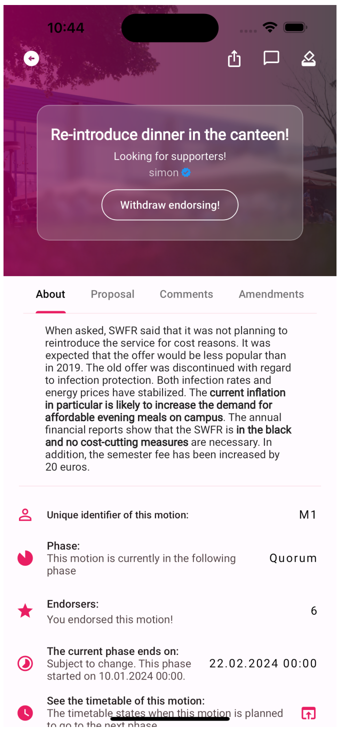
Phase:
This motion is currently in the following phase
Deliberation

Endorsers:
You endorsed this motion!
1

The current phase ends on:
Subject to change. This phase started on 01.02.2024 00:00.
04.02.2024 00:00

See the timetable of this motion:

(b) Motion in deliberation phase.


10:44

Re-introduce dinner in the canteen!
Looking for supporters!
simon
Withdraw endorsing!

About Proposal Comments Amendments

When asked, SWFR said that it was not planning to reintroduce the service for cost reasons. It was expected that the offer would be less popular than in 2019. The old offer was discontinued with regard to infection protection. Both infection rates and energy prices have stabilized. The **current inflation in particular is likely to increase the demand for affordable evening meals on campus**. The annual financial reports show that the SWFR is **in the black and no cost-cutting measures** are necessary. In addition, the semester fee has been increased by 20 euros.

Unique identifier of this motion: M1

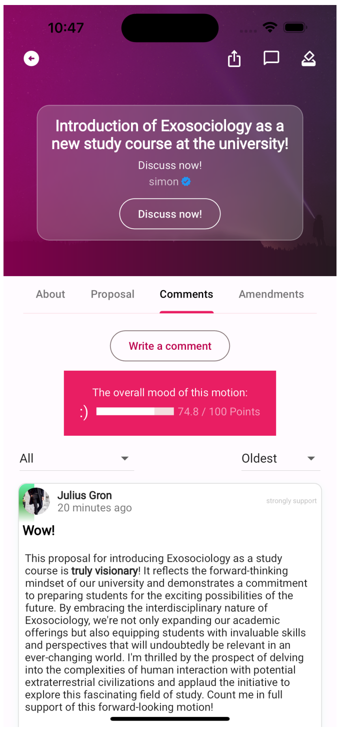
Phase:
This motion is currently in the following phase
Quorum

Endorsers:
You endorsed this motion!
6

The current phase ends on:
Subject to change. This phase started on 10.01.2024 00:00.
22.02.2024 00:00

See the timetable of this motion:
The timetable states when this motion is planned to go to the next phase

(c) An endorsed motion in the quorum phase.


10:47

Introduction of Exosociology as a new study course at the university!
Discuss now!
simon
Discuss now!

About Proposal Comments Amendments

Write a comment

The overall mood of this motion:
:) 74.8 / 100 Points

All Oldest

Julius Gron
20 minutes ago
strongly support

Wow!

This proposal for introducing Exosociology as a study course is **truly visionary**! It reflects the forward-thinking mindset of our university and demonstrates a commitment to preparing students for the exciting possibilities of the future. By embracing the interdisciplinary nature of Exosociology, we're not only expanding our academic offerings but also equipping students with invaluable skills and perspectives that will undoubtedly be relevant in an ever-changing world. I'm thrilled by the prospect of delving into the complexities of human interaction with potential extraterrestrial civilizations and applaud the initiative to explore this fascinating field of study. Count me in full support of this forward-looking motion!

(d) The discussion section of a motion.

Figure 3: Different screens of the Polar client on an Apple iPhone 15 Pro Max.

Polar calculates an overall mood for the motion out of these top-level comment moods, also respecting the amount of likes each comment gets (this is possible as Polar accounts are linked to one user only, meaning there is no distortion through bots). For more on this, check out Chapter 4.3.2. During the discussion phase, a motion can also collect amendments, and these amendments contain a proposal (what to change of the motion) and a description as well. Another student could, in our example, submit an amendment which adds a passage to the proposal of the motion that the canteen should always offer vegan options in the evening.

5. **Post-deliberation validation phase**: Once the discussion phase is over, the motion council can edit or group amendments (respecting the rules of the organisation), conduct prior negotiations between amendment applicants to summarize them with the aim to reduce the total amount of amendments. During this phase, the motion council is presented with tools to automatically create the election on the integrated election service (Eos).

6. **Negotiation phase**: the negotiation phase exists to allow members to vote on the adoption of the amendments through Eos.

7. **Post-negotiation validation phase**: The motion council checks the results posted on Eos and adapts the proposal of the motion with all adopted amendments. Polar provides tools to automatically create the final election on Eos.

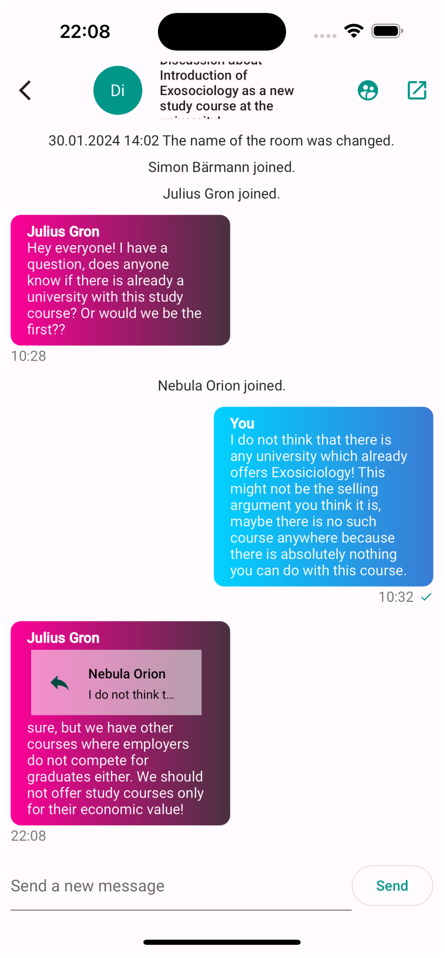
8. **Election phase**: The members can vote on the adoption of the Motion on Eos. Once the election results are available, the motion is either adopted or rejected. If the motion is adopted, the student council would then release the motion's proposal as an resolution.

Each of these phases can take only a few days up to a few weeks, especially the discussion phase can be set quite long to allow a rich and diverse discussion to happen.

The motion process can also be capped at a different stage to, for example, use Polar as a motion collection tool. This can be achieved by disabling the deliberation, negotiation and election phase in the respective MotionRules, turning Polar into an addition to an offline workflow (like a real party convention).

3.1.2 Chats

To enable members to discuss on topics outside of the formal process of motions, Polar enables chat rooms. Polar uses a Matrix [16] server for this. Matrix is a chat protocol which allows “an open network for secure, decentralised communication”. In the context of Polar, the focus is not mainly on the decentralised aspect of Matrix. While Polar users can chat with users from other Matrix homeservers (like matrix.org or other Polar instances), Matrix was mainly chosen because it allows encrypted chat rooms, supports various open-source clients and is reliable and well-tested. To connect a Matrix server (like Synapse [17]) with Polar, the Matrix server needs to support OpenID-Single-Sign-On systems. Furthermore, the Polar server manipulates Matrix only in that way that it creates new Matrix rooms, all other chat related features are completely managed by the Matrix server. So, as long as the Matrix server supports OpenID-Single-Sign-On and the creation of (private) rooms with an alias, Polar is compatible with a big variation of Matrix



30.01.2024 14:02 The name of the room was changed.

Simon Bärmann joined.

Julius Gron joined.

Julius Gron
Hey everyone! I have a question, does anyone know if there is already a university with this study course? Or would we be the first??
10:28

Nebula Orion joined.

You
I do not think that there is any university which already offers Exosiciology! This might not be the selling argument you think it is, maybe there is no such course anywhere because there is absolutely nothing you can do with this course.
10:32

Julius Gron
Nebula Orion
I do not think t...
sure, but we have other courses where employers do not compete for graduates either. We should not offer study courses only for their economic value!
22:08

Figure 4: A chat group on a motion.

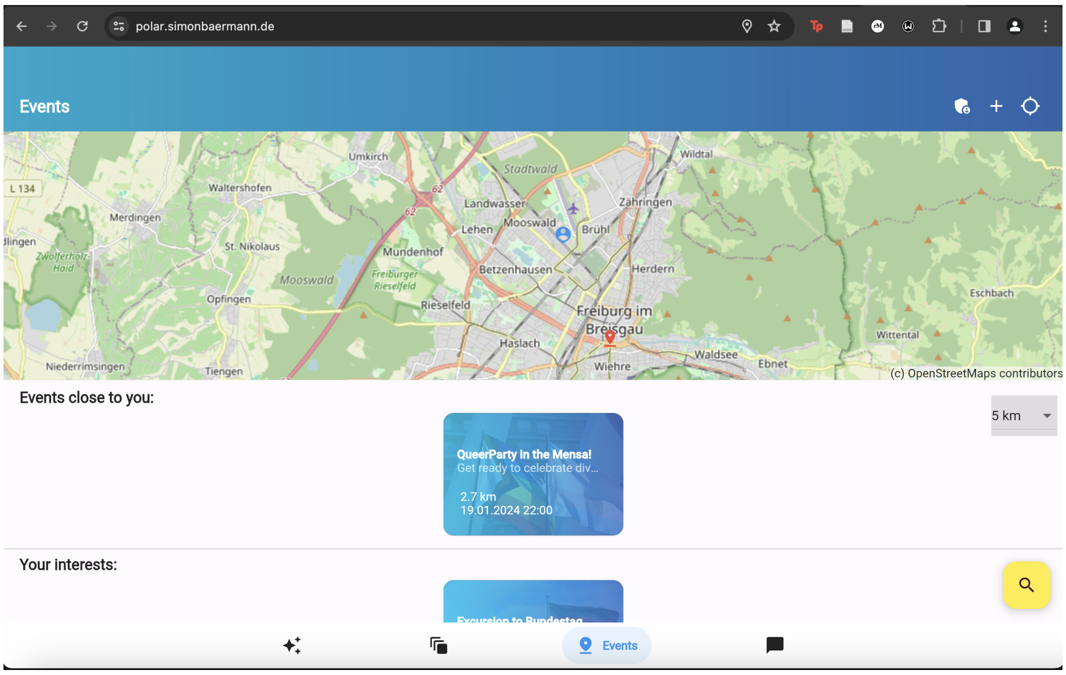
servers.

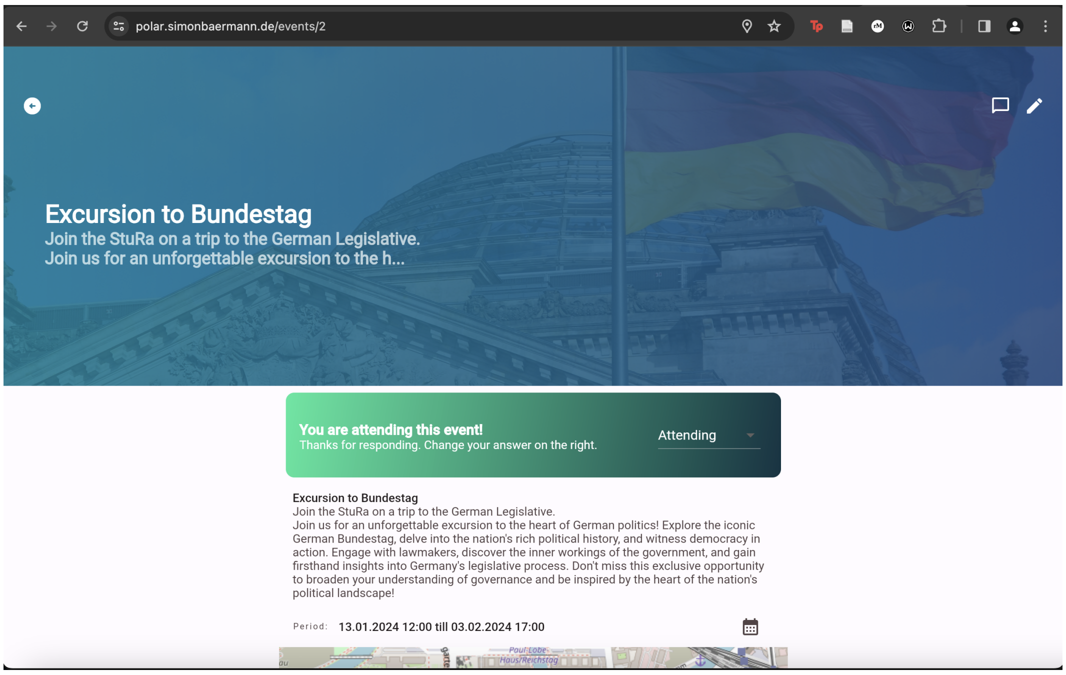
The Polar client supports the basic chat features (as can be seen in Figure 4). Members can join and leave chat rooms, create private chats and accept (or reject) invitations. They can also see images and receive files as well as use the Matrix 'respond to' feature (up to a depth of one). The Polar client does not support sending non-text messages of any kind, does not support Voice-over-IP calls nor displays which user has read which message (while it does send read receipts). Users can very easily switch to a full-weight Matrix client like Element [18] and use this app to chat with other members. The Polar client chat feature therefore is a proof-of-concept, but it would be out of the scope of this work to fully support all Matrix features.

Polar creates chat rooms for all motions and events, allowing an informal discussion place for its members. This can be understood as the ground-floor of a political convention. These chat rooms are not encrypted, but Polar sets the correct flag to prevent people from other Matrix homeservers to join the room (disallowing any non-Polar members to access the chat room).

3.1.3 Events

Every (political) group has events it organizes or invites its members to join. Currently, invitations to these events are spread to the members by email, chat groups or with the Facebook event tool. Members that are, for any reason, not part of the email or chat group, or members that don't want to use Facebook or any other third party social media service may miss out on important group-related information. That is why Polar has an Event feature which enables members (that were granted the right to do so) to create and share events (as can be seen in Figure 5). These events contain a position or a link to a website like Zoom (or both), a description and a timeframe. Members can then comment on the event. They can also respond, whether they are interested, commit to attend or state that they will not. The Polar client asks to




polar.simonbaermann.de/events/2
Excursion to Bundestag
Join the StuRa on a trip to the German Legislative.
Join us for an unforgettable excursion to the h...
You are attending this event!
Thanks for responding. Change your answer on the right.
Attending

Excursion to Bundestag
Join the StuRa on a trip to the German Legislative.
Join us for an unforgettable excursion to the heart of German politics! Explore the iconic German Bundestag, delve into the nation's rich political history, and witness democracy in action. Engage with lawmakers, discover the inner workings of the government, and gain firsthand insights into Germany's legislative process. Don't miss this exclusive opportunity to broaden your understanding of governance and be inspired by the heart of the nation's political landscape!

Period: 13.01.2024 12:00 till 03.02.2024 17:00

Figure 5: *Top*: The events feed page showing events close to the user. *Bottom*: A single page of an event. The Polar web client is accessed using Google Chrome on an Apple MacBook Pro.

access the current position of the device, if the user allows this, it sends this location to the server. The server responds with the closest events around, as explained in Chapter 4.3.1.

3.1.4 Single-Sign-On Provider (SSO)

The Polar server, as already stated before, can be used as a Single-Sign-On (SSO) service. The OAuth2AuthenticationFlow is implemented on a basic level, allowing the integration of Polar to other services, which support SSO. This was tested with Eos and Synapse (the Matrix server) but one could also imagine other integrations with services like NextCloud.

3.1.5 Elections

Polar can not be used to conduct elections, but you can connect it with Eos (or any other election service, for which you have to implement a new connector class). Elections, that are created with the Polar admin account on Eos are featured to the members on the Polar client start page. Also, every member can vote on the elections on Eos with their Polar account, they do not need to create a second account, even with Eos and Polar being two independent software systems.

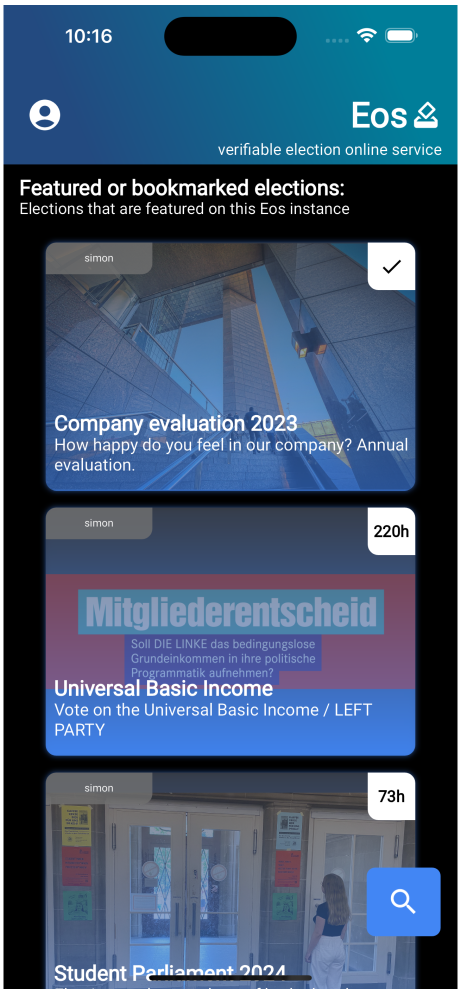
Eos allows to outsource the eligibility check. Polar uses this, and therefore Polar conducts the eligibility check for the two elections on motions. This means, Polar will not send a list of the eligible members to Eos.

These are the main features of Polar. It also supports uploading images and a user management system, with different user roles defining what each user of this role can or can not do.

Figure 6: The Eos logo.

3.2 Eos

Eos is a software allowing encrypted digital elections. The main focus in the conception of Eos was to provide an understandable and verifiable en- and decryption process. Eos (Figure 7) supports the creation of elections by any user (see Chapter 6.2.2 for a comment on this). Elections contain a ballot, which consists of questions. These questions encapsulate answer options. The creator can set for each question how many votes a voter has as a minimum and as a maximum. The creator can also set for each answer option how many votes this answer can get as a maximum. This allows questions to be either of ’only one answer’ type (for example if the election is on the adoption of a motion) or make it possible to cumulate and split votes as a voter (useful, for example, when it comes to elections on political personnel, as can be seen in Figure 8d). Eos supports eligibility checks by providing the user id, the email or the username of a user, and also a third-party can be added as a eligibility check system. Voters can fully register or use a one-time code which is sent to their e-mail to authenticate themselves. Furthermore they



Featured or bookmarked elections:
Elections that are featured on this Eos instance

Company evaluation 2023
How happy do you feel in our company? Annual evaluation.

Soll DIE LINKE das bedingungslose Grundeinkommen in ihre politische Programmatik aufnehmen?

Universal Basic Income
Vote on the Universal Basic Income / LEFT PARTY

Student Parliament 2024

Figure 7: The main page of Eos.

can login with any third-party SSO system that is setup on the server (see Chapter 4.2.3 on this). Once the admin sets the timeframe in which the election happens, the Eos client will show a countdown. Once the election has started, voters can access the ballot, verify it and cast their vote. They are only asked to authenticate after they chose to cast it. Once their eligibility was proven and their encrypted ballot was uploaded, they receive an alias name for this election, under which the ballot was cast. Also, before submitting, the client already displays the hash of the encrypted vote. Everyone who has access to the election can always access all encrypted ballots in the tally and make sure that the hash and the actual encrypted object have not changed (as can be seen in Figure 8a and 8b). This means, the Eos server did not swap votes (see Chapter 6.1.1 for further ideas on this).
Neither Eos nor Polar support the delegation of votes. Each vote has always the same weight (1 out of v where v is the amount of total voters).
Helios [5] offers various interesting concepts for auditable online elections. But the homomorphic encryption is complex and zero-knowledge-proofs not really understandable for non-experts. The code is also quite old and poorly documented. It was not possible for me to actually verify my encrypted vote using the tools Helios provides. Further, Helios is not able to allow different amounts of allowed votes for answers to the same question of the same election, a problem political groups and organizations usually face. Therefore, Eos was designed inspired by many of the important design choices made by Helios. Eos also allows anyone to encrypt a ballot and verify the encryption (as can be seen in Figure 8e), if the election was not marked as private by the author. Eos also gives the voter a 'fingerprint'-like combination of an generated alias and a hash to verify their cast vote in the publicly available cast ballot list (displayed in Figure 8f). Eos supports trustees, just like Helios introduced them in a later version. But Eos uses a different encryption protocol (which is much easier to understand and verify than homomorphic encryption), does not keep a direct link between cast vote and user account, and Eos decrypts and publishes each cast ballot. Helios only publishes zero-knowledge proofs, which, arguable, would not actually

convince non-experts that their vote was respected in the result. This is why the data structure of Eos is similar to that of Helios, but Eos does not use Helios code, nor are the systems compatible with each other. To value the important ideas Helios established and the connection between the two apps, the name Eos was chosen. While Helios is the ancient Greek god of the sun, Eos is his sister, the goddess of the dawn.

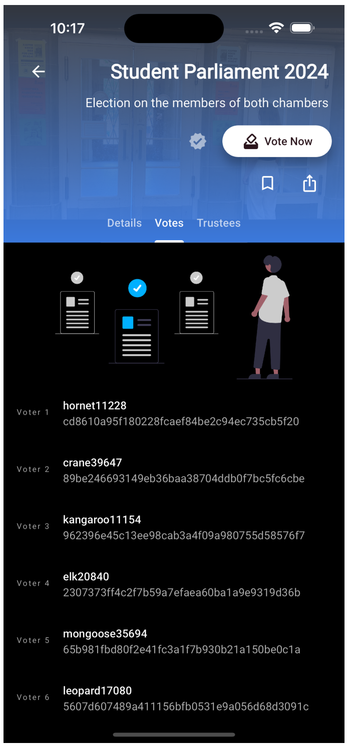
3.2.1 Encryption and Trustees

Eos supports third-party trustees. This assures that the Eos server does not decrypt the votes before the election is over. It also ensures that no participant in the process manipulates the election results. Each trustee has for each registered decryption job, an object which tells a trustee which elections it is part of, a public and a private RSA key. RSA [19] is an asymmetric encryption protocol. Asymmetric encryption protocols consist of two different keys. A public key used to encrypt content and a private key used for decryption. The Eos server acts as a trustee as well, though it has a different role than third-party trustee, as the Eos server orchestrates the decryption process between the different trustees.

The Eos client encrypts the vote of the user by generating and using an AES [20] (a symmetric encryption protocol) secret. Symmetric encryption protocol consist of a single secret used for both en- as well as decryption. This secret then is encrypted by using the public key of the current trustee. This is done multiple times, starting with the Eos server trustee, up to the latest trustee.

For the decryption, Eos shares the encrypted votes to the corresponding trustee, which will decrypt and return the 'partly-decrypted' votes. The trustees will be called on in the reversed order of the encryption till only the layer where the secret is encrypted with the Eos public key is left. Eos will then finally decrypt the votes and publish the results of the election. Users can then see the results, but also see the encrypted and decrypted vote by any user. If they remember their alias, they can make sure this equals the vote they cast (even with totally corrupt trustees and servers, only one vigilant voter is necessary to raise suspicion of a tampered vote). The detailed explanation on the en- and decryption protocol can be found in Chapter 4.2.1.

The Eos trustee server published together with the Eos client and the Eos server is written in Python3.11 and uses FastAPI [1] and SQLAlchemy [2].

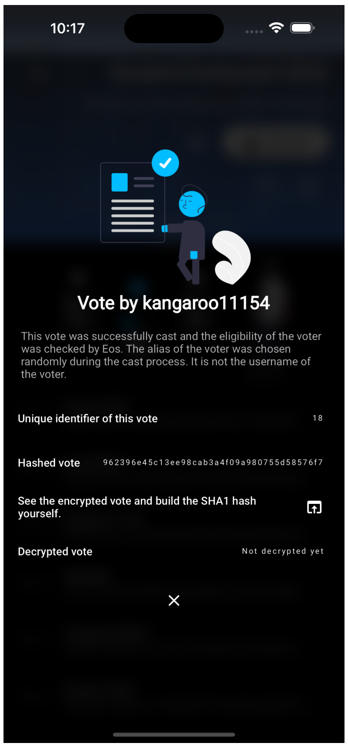


Student Parliament 2024

Election on the members of both chambers

Voter 1	hornet11228 cd8610a95f180228fcaef84be2c94ec735cb5f20
Voter 2	crane39647 89be246693149eb36baa38704ddb0f7bc5fc6cbe
Voter 3	kangaroo11154 962396e45c13ee98cab3a4f09a980755d58576f7
Voter 4	elk20840 2307373ff4c2f7b59a7efaea60ba1a9e9319d36b
Voter 5	mongoose35694 65b981fbd80f2e41fc3a1f7b930b21a150be0c1a
Voter 6	leopard17080 5607d607489a411156bfb0531e9a056d68d3091c

(a) See all cast votes

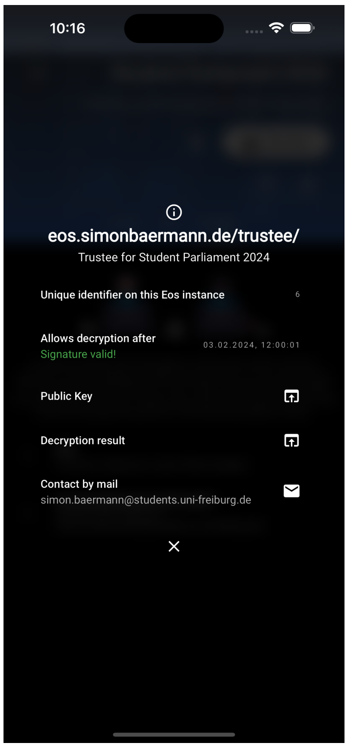


Vote by kangaroo11154

This vote was successfully cast and the eligibility of the voter was checked by Eos. The alias of the voter was chosen randomly during the cast process. It is not the username of the voter.

Unique identifier of this vote	18
Hashed vote	962396e45c13ee98cab3a4f09a980755d58576f7
See the encrypted vote and build the SHA1 hash yourself.	
Decrypted vote	Not decrypted yet

(b) Access any cast vote

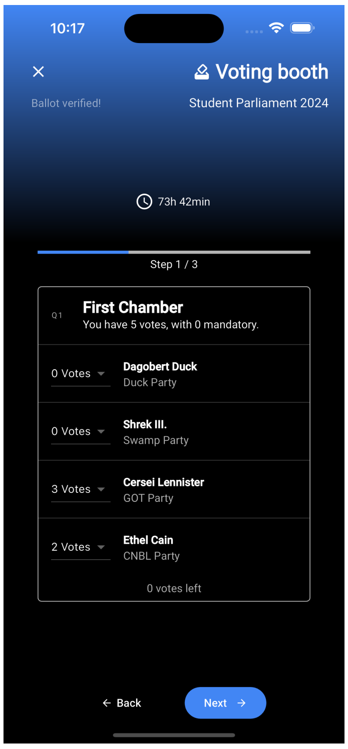


eos.simonbaermann.de/trustee/

Trustee for Student Parliament 2024

Unique identifier on this Eos instance	6
Allows decryption after Signature valid!	03.02.2024, 12:00:01
Public Key	
Decryption result	
Contact by mail simon.baermann@students.uni-freiburg.de	

(c) Analyze trustees



Ballot verified!

Student Parliament 2024

73h 42min

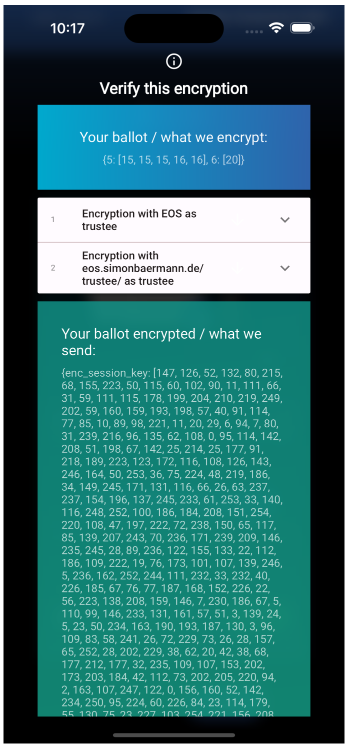
Step 1 / 3

Q1 **First Chamber**
You have 5 votes, with 0 mandatory.

0 Votes	Dagobert Duck Duck Party
0 Votes	Shrek III. Swamp Party
3 Votes	Cersei Lennister GOT Party
2 Votes	Ethel Cain CNBL Party

0 votes left

(d) Vote in the election



Verify this encryption

Your ballot / what we encrypt:
{5: [15, 15, 15, 16, 16], 6: [20]}

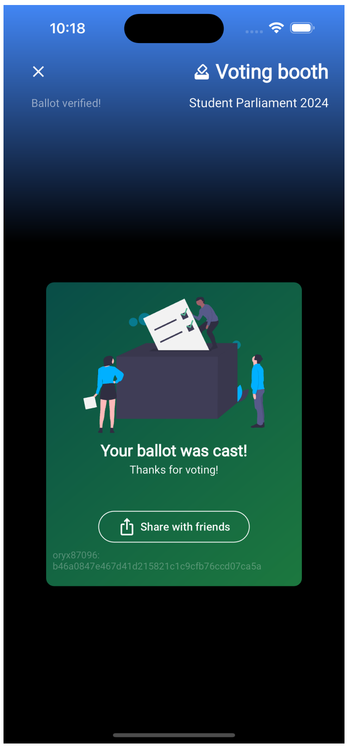
1 Encryption with EOS as trustee

2 Encryption with eos.simonbaermann.de/trustee/ as trustee

Your ballot encrypted / what we send:

```
{enc_session_key: [147, 126, 52, 132, 80, 215,
68, 155, 223, 50, 115, 60, 102, 90, 11, 111, 66,
31, 59, 111, 115, 178, 199, 204, 210, 219, 249,
202, 59, 160, 159, 193, 198, 57, 40, 91, 114,
77, 85, 10, 89, 98, 221, 11, 20, 29, 6, 94, 7, 80,
31, 239, 216, 96, 135, 62, 108, 0, 95, 114, 142,
208, 51, 198, 67, 142, 25, 214, 25, 177, 91,
218, 189, 223, 123, 172, 116, 108, 126, 143,
246, 164, 50, 253, 36, 75, 224, 48, 219, 186,
34, 149, 245, 171, 131, 116, 66, 26, 63, 237,
237, 154, 196, 137, 245, 233, 61, 253, 33, 140,
116, 248, 252, 100, 186, 184, 208, 151, 254,
220, 108, 47, 197, 222, 72, 238, 150, 65, 117,
85, 139, 207, 243, 70, 236, 171, 239, 209, 146,
235, 245, 28, 89, 236, 122, 155, 133, 22, 112,
186, 109, 222, 19, 76, 173, 101, 107, 139, 246,
5, 236, 162, 252, 244, 111, 232, 33, 232, 40,
226, 185, 67, 76, 77, 187, 168, 152, 226, 22,
56, 223, 138, 208, 159, 146, 7, 230, 186, 67, 5,
110, 99, 146, 233, 131, 161, 57, 51, 3, 139, 24,
5, 23, 50, 234, 163, 190, 193, 187, 130, 3, 96,
109, 83, 58, 241, 26, 72, 229, 73, 26, 28, 157,
65, 252, 28, 202, 229, 38, 62, 20, 42, 38, 68,
177, 212, 177, 32, 235, 109, 107, 153, 202,
173, 203, 184, 42, 112, 73, 202, 205, 220, 94,
2, 163, 107, 247, 122, 0, 156, 160, 52, 142,
234, 250, 95, 224, 60, 226, 84, 23, 114, 179,
55, 130, 75, 23, 227, 103, 254, 221, 156, 208
```

(e) Verify the encryption



Ballot verified!

Student Parliament 2024

Your ballot was cast!

Thanks for voting!

oryx87096:
b46a0847e467d41d215821c1c9cfb76ccd07ca5a

(f) Cast successfully

Figure 8: Further screenshots of Eos on an Apple iPhone 15 Pro Max.

4 Theoretical Analysis

In this chapter the focus is on the overall architecture of both apps as well as on particularly relevant algorithms.

4.1 The Architecture

Both Eos and Polar use a client-server architecture. The clients (Polar client and Eos client) communicate by using the internet with the servers (Polar server and Eos server). Usually, the amount of clients vastly exceeds the amount of servers, as one server can satisfy the needs of many clients. The communication protocol used for the communication between the client and the server uses the Representational State Transfer (REST) [21] architecture, widely used in the World Wide Web. It is also very easy to debug and easy to understand for third-party experts that want to analyze the app. Also, it is rather easy to create a third-party inofficial client. This is especially interesting for Eos, so users do not have to trust the official client and can use a third-party or their own client to vote on elections. The subtleties of the REST architecture are handled by the 'FastAPI' [1] framework which is used by both server applications. FastAPI allows the developer to focus on working on their software-specific aspects, while the framework takes over all communication-specific tasks. With SQLAlchemy [2], an object-relation mapping framework which converts the database objects into objects in the object-orientated language (Python), the servers communicate with the relational database which stores the data of the

applications. SQLAlchemy converts the function calls in Python code to SQL code, which adds a heap of a few milliseconds on every call, but this is rather neglectable as SQLAlchemy caches calls that were already converted. SQLAlchemy sanitizes SQL calls so that the evil username ';DROP USERS;' does not actually drop the users table.
The clients are made in Flutter [4], which is an open-source framework for creating multi-platform applications on a single codebase. With Flutter, developers code almost the full app in Dart. Only platform-specific code needs to be developed per platform, which was not necessary for neither Polar nor Eos. Flutter supports Android, iOS, Web, Windows, macOS and Linux as platforms. Both the Polar and Eos clients can run on all these platforms, but are only tested on Android, iOS and the Web. The native Polar client can not acquire the current position of the user on Linux, macOS or Windows. This limitation is based on the package which provides Polar with the location of the user.

4.2 Eos

4.2.1 En- and Decryption Protocol

The en- and decryption of votes is the crucial feature of Eos. The protocol was designed to make the verification as easy to understand as possible. This inevitably means that cuts had to be made elsewhere. Encryption procedures for online elections must always decide which factor they prioritize: comprehensibility/verifiability or privacy protection. In the current protocol, it is possible that a corrupt Eos server maps decrypted votes to their user and thus breaks the principle of secret ballot in retrospect. However, unless all trustees are corrupt, it is impossible even for the corrupt Eos server to decrypt the votes before the end of the election. Arguably, even voting procedures such as Helios, whose encryption methods are more complicated and whose verifiability cannot be understood by the non-expert, can ultimately enable such a breach of electoral secrecy in the case of a corrupt Helios instance. Even homomorphic encryption protocols do not stop a server from decrypting all votes individually, if the server has all the secrets necessary for decryption. A voter must therefore, to a certain extent, always trust the respective election server. This is also the case for offline elections, where voters have to trust the organizing party up to some extent. However, Eos presents the voters with all trustees who are registered for the respective election. Voters can therefore see for themselves how trustworthy they think all participating trustees of the election are (as can be seen in Figure 8c).

There is a distinction between Voter and User. While a Voter and a User can be the same real-life entity, the Eos server tries to make sure that a Voter and a User object can not be linked. $Users$ are Eos objects, which can login. Users are on eligibility systems, they can create elections and users can, using their sensitive user data (like email), be traced back to real people. Voters are actually just cast votes (CastVote objects). The voter is the alias this cast vote got, generated by the server the moment the user submits their encrypted ballot. The generation works by Eos choosing a random name of an animal and adding a random 5-digit long number to the name (see Chapter 6.2.3 for a comment on this). To ensure that no user votes twice, we map the user id to the corresponding election id they voted on. However, no map is created that links which user is which voter/cast vote.
After the user completed to answer all questions on the ballot, the client encrypts the vote. See also Algorithm 1 and Figure 10. First, the client collects all public keys $r_0, ..., r_n$ from all trustees n for this election, where r_0 is the public key of the Eos server trustee, sorting the rest ascending by the unique trustee id, so we get $r_1, ..., r_n$. This sorting is important as a vote that is encrypted in a different order can not be decrypted in the current protocol (while it is possible to adapt the protocol to support this). The client then loops through the public keys, encrypting the current layer x which is in the beginning the ballot as a map, then the result of the encryption of the previous layer as a map.

To encrypt, the client generates an AES secret a_i. AES is a symmetric encryption protocol, meaning that this secret is used to encrypt and decrypt data. We use a_i to actually en- and decrypt x. Then we encrypt this secret with the current RSA public key r_i. RSA is an asymmetric encryption protocol, this means that the public key can be used to encrypt data, but can not be used to decrypt the data. Only those who have access to the private key can decrypt the data, hence why it should stay private. Once we have encrypted x with a_i, receiving a ciphertext and the AES nonce, and encrypted a_i receiving $encA_i$ we create a map with $encA_i$, *ciphertext*, *nonce* and set x to this map. Then we continue this loop till we reached the last trustee.

See Table 1 for benchmark results of this algorithm. The runtime of the encryption algorithm is dependent on the amount of trustees n and the runtime of the implementation of the RSA encryption T_{rsa} and the AES encryption T_{aes} protocol, which both is in $\Theta(1)$. The runtime of T_{enc} is therefore in $\Theta(n)$. In practice, on a modern CPU, the encryption runtime is fast even with a lot of encryption layers (trustees). The size of the encrypted ballot grows exponentially with the amount of trustees n used for the encryption. An alteration of the encryption protocol could only generate one AES secret a and then encrypt this multiple times instead of the whole ballot. This would reduce the overall size of the encrypted ballot but would also increase the risk of a brute force attack on the cast votes.
Once the encryption is done, the user can verify the encryption. This displays $a_0, ..., a_n$ which the client saves till the user presses the cast button so the user can use any encryption software to verify the result is correct. Also, the user sees the final encryption layer x and can calculate the hash of this and verify that the hash the Eos client presents to the user is correct.
Pressing cast, the user has to authenticate or confirm that they want to cast their vote with the currently logged in account. Users cannot revoke their vote, therefore we ask for confirmation here. Once the user has confirmed, the server checks the registered eligibility systems (ES) for this election whether the user is eligible. An Eos election

can have a id/username-system ES, an email ES and or multiple third-party ES. If the logged in user is not part of the id/username or email system, the third-party eligibility systems are asked whether the currently logged in user is eligible. The Eos server sends a minimal set of information to the third-party, to make sure the third-party system does not receive sensitive information. As users can always check their eligibility, the third-party system also can not be sure if the user is actually casting their vote right now. Once one of these eligibility systems reacts positively, the vote is cast and saved in the database, the user is added to the map that links user ids with the election ids they cast a vote on and the client receives the alias name and a positive response. The cast process is then finished.

Trustees	Runtime (ms)	Encrypted Ballot Size (bytes)
1	10*	1478
2	2	6723
3	5	25551
4	5	93399
5	8	328095
6	17	1170009
7	47	4200842
8	161	14906195

Table 1: Benchmark results of the Eos encryption algorithm. This table displays the runtime and the encrypted ballot size for an Election with the given size of trustees. The ballot consists of two questions with each 4 selected answers. Runtime on an Apple MacBook Pro with M1 chip, 16GB of RAM, RSA keysize was 2048.
*Encryption with one trustee seems to take longer than with several trustees. The reason behind this is that CPU optimizations took into effect after the operating system recognized that it was presented the same task (to encrypt) multiple times.

Everyone who has access to the election can access all cast votes at any time, they are downloadable and once the decryption is done, also the decrypted ballot is available. Users and third-party watchers, like experts or security analysts, can always verify

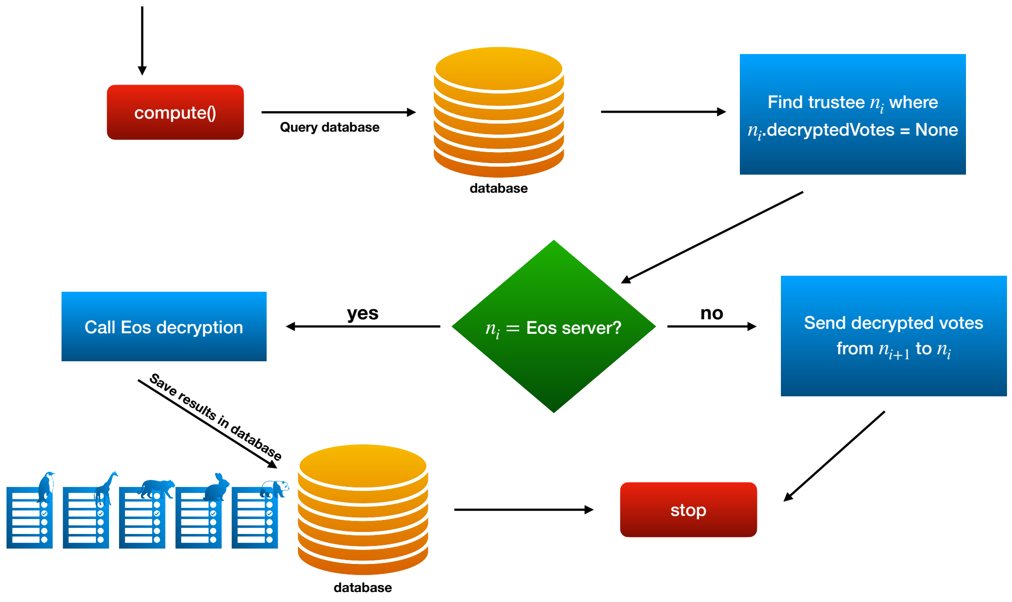


Figure 9: Compute is the main method which orchestrates the decryption process. The Eos server has this method. Compute is called once the election is over and whenever a trustee delivers their decryption result. Also, if a trustee is not reachable, Eos will try calling the trustee again after some time. Compute is therefore an asynchronous function. It does not wait for an immediate answer of the trustee.

that during the voting no one swaps votes nor that a vote vanishes. Also, the result map of Eos is easily verifiable by simply calculating it by adding all decrypted vote results. Trustees provide their decryption result object, which is an incomplete decryption result, as an api call as well. The Eos client can display this result, so that experts can use this to verify the decryption process.

The Eos decryption process is orchestrated centrally by the Eos server. See Figure 9, Figure 11 and Algorithm 2. The method compute is called either by api request, by the scheduler once the election is over and whenever a trustee delivers their decryption result. The Eos server keeps track which trustee has already delivered their decrypted data and provides the current decryption layer to the next trustee. Once the Eos server is the only remaining encryption layer, the Eos server finally decrypts the

votes, which are now the ballots, counts them, and publishes the tally. The Eos server verifies that the votes fulfill the constraints of the questions and answers of the election (neither exceeding nor subceeding the max and min votes constraints). It does not check the eligibility again, as that is not possible because there is no map linking user and cast vote. Once the results are published, they are immediately available to users in the client. The runtime of the trustee decryption process is dependent on the amount of votes v and the runtime of the RSA decryption and the AES decryption protocol (both in $\Theta(1)$). See Table 2 for runtime details on the decryption process of trustees. While the final decryption process of Eos is also dependent on the amounts of questions q the election has.
The runtimes are $T_{trusteedec} = v$ and $T_{eosdec} = T_{trusteedec} + v \cdot q = 2vq$. The complete decryption process then is $T_{dec} = (n-1) \cdot v + 2vq$. Following, $T_{dec} \in \Theta(n)$.

Layers	Runtime (ms)
1	5.762
2	3.37
3	3.526
4	3.606
5	5.336
6	9.175
7	20.71
8	66.868

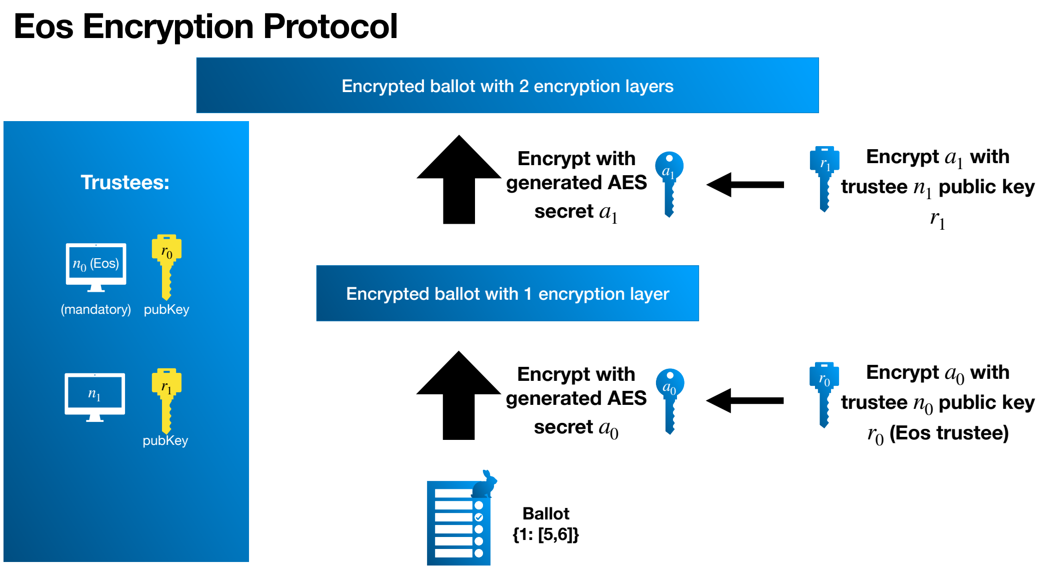
Table 2: Benchmark results of the Eos decryption. This table displays the runtime it takes the Eos trustee to decrypt one layer of one vote, see in Table 1 what size the vote is. Runtime on an Apple MacBook Pro with M1 chip, 16GB of RAM.
RSA keysize was 2048

Algorithm 1 Eos Encryption Algorithm

```
Let $r_1, ..., r_n$ be the RSA public keys of the trustees.
Let $r_0$ be the RSA public key of the Eos server.
Let x be the ballot as a json map.
foreach pubKey $\mathbf{r_i}$ do
    $x \leftarrow$ toUTF8(x)                                  ▷ Convert ballot to utf8
    $a_i \leftarrow$ aes.generateSecret()                     ▷ Get AES secret for this round
    $ciphertext, nonce \leftarrow$ aes.encrypt($\mathbf{x, a_i}$)    ▷ Encrypt x with AES
    $encA_i \leftarrow$ rsa.encrypt($\mathbf{a_i, r_i}$)            ▷ encrypt secret with RSA
    $x \leftarrow$ map($\mathbf{encA_i}$, ciphertext, nonce)       ▷ Set x to a json map of en-
                                                              crypted prev x and encrypted
                                                              secret key
end for
```

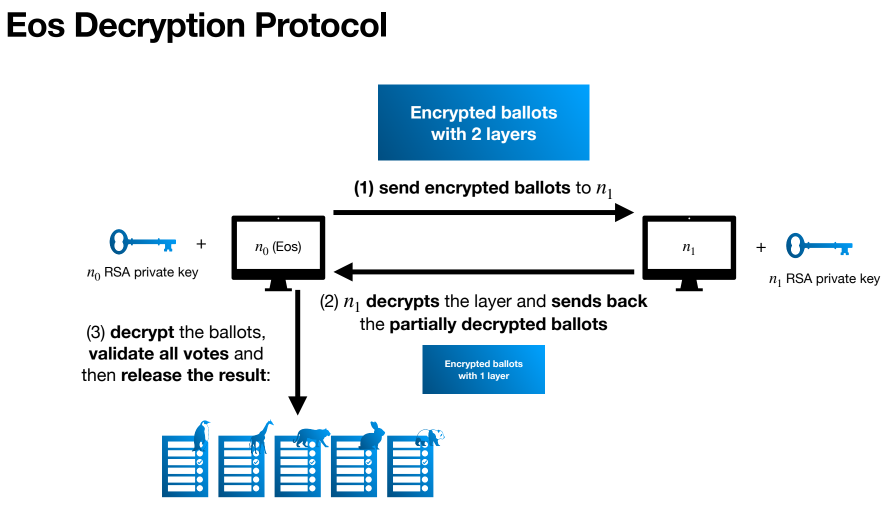
Algorithm 2 Eos Decryption Algorithm

```
Let $decVotes$ be an empty map that stores the id of the question as keys and saves
a list of selected answer ids as value.
Let $privKey$ be the Eos private key for this election.
foreach vote $\mathbf{v_i}$ do
    $secret \leftarrow$ rsa.decrypt($\mathbf{v_i.encA_i}$, privKey)      ▷ Decrypt the AES secret
    $data \leftarrow$ aes.decrypt($\mathbf{v_i}$.ciphertext, secret)    ▷ Decrypt the data of this vote
    $vote \leftarrow$ fromUTF8(data)                              ▷ Decode data encoded as utf8
    foreach question q in $vote.keys$ do
        $ans \leftarrow$ vote[q]                                  ▷ This is a list of answer ids
                                                              the voter has chosen for the
                                                              current question.
        $decVotes[q].append(ans)$                                 ▷ append the answers for this
                                                              question to the decrypted
                                                              votes list
    end for
    $v_i.decryptedVote \leftarrow$ vote                            ▷ Publish decryption result
end for
$result \leftarrow$ calculateResult(decVotes)                      ▷ Calculate results for each
                                                              question 'qid': ['aid': [nu-
                                                              mOfVotes, relResult]].
```



Eos Encryption Protocol
Encrypted ballot with 2 encryption layers
Trustees:
n_0 (Eos)
r_0
(mandatory)
pubKey
n_1
r_1
pubKey
Encrypt with generated AES secret a_1
a_1
Encrypt a_1 with trustee n_1 public key r_1
Encrypted ballot with 1 encryption layer
Encrypt with generated AES secret a_0
a_0
Encrypt a_0 with trustee n_0 public key r_0 (Eos trustee)
Ballot {1: [5,6]}

Figure 10: The Eos encryption protocol using two trustees.



Eos Decryption Protocol
Encrypted ballots with 2 layers
(1) send encrypted ballots to n_1
n_0 RSA private key
+
n_0 (Eos)
n_1
+
n_1 RSA private key
(2) n_1 decrypts the layer and sends back the partially decrypted ballots
Encrypted ballots with 1 layer
(3) decrypt the ballots, validate all votes and then release the result:

Figure 11: The Eos decryption protocol using two trustees.

4.2.2 Setup Trustee Protocol

Every election e has n trustees where $n \geq 1$ must hold. This first trustee n_0 is always the Eos server on which e resides. To add another trustee, either the client communicates with both the Eos server and the third-party trustee server and coordinates the setup or the client transfers this setup to the Eos server which then directly communicates with the trustee server. The result of this protocol is a new decryption job object on the trustee server, which consists of a public and private RSA key, where the public key is available to download at any time. Further, the trustee has a decryption time object $decTime$, which is the earliest time on which the trustee allows incoming votes to be decrypted. $decTime$ is always available to download and signed with the RSA private key. Also, the public key of n_0 is saved on the trustee as is the IP address of the Eos server. $decTime$ can be changed by and only by the Eos server, but the trustee will check if the sent new $decTime$ equals the time the election ends on the server. To deliver the encryption results to the Eos server, the trustee also gets a user-like object on the Eos server, with which it authenticates itself. Still, the decryption results are also signed with the trustee's private key. Only when the password is correct, the signature of the decryption results are validated by the Eos server. This ensures that the server is not flooded with big maps of falsely signed decryption results, where the validation is abused to slow down the server. The signature ensures that third-party watchers can be sure that the decryption result is from the trustee without the leak of the password. The protocol works as follows:

1. The user provides the Eos client with a jwtToken k with which the Eos client can authenticate to the new trustee server t.

2. The Eos client generates a secure password s which will be used for the trustee account on the Eos server.

3. In this example, the client transfers the setup to the Eos server and waits for the final confirmation.

4. The Eos server asks t for information about the trustee server. Receives $name$ and $email$ of the trustee server.

5. The Eos server creates a new internal trustee object n_i which receives an unique eos id $uei(n_i)$ and the server creates the respective trustee user account using s.

6. The Eos server creates a new decryption job object with the previously acquired information and sends this to t.

7. If successful, t returns an internal job id for this newly created decryption job and its public key.

8. The Eos server updates its internal n_i object. Now the trustee is successfully registered. The Eos server confirms the Eos client that the setup was successful.

4.2.3 AuthPartners

Eos allows admins to add third-party authentication partners, with which users can authenticate themselves, as can be seen in Figure 13. A precondition is that the AuthPartner supports OAuth2-Authentication-Auth-Flow, an open standard for authentication between two services, where a user with an account at service A wants to authenticate with this account at service B. The idea of this authentication flow is that service B and the client of service B do not get the password of the user account for A. In our case, Eos takes the role of service B, while the third-party authentication partner is service A. This is pictured in Figure 12.

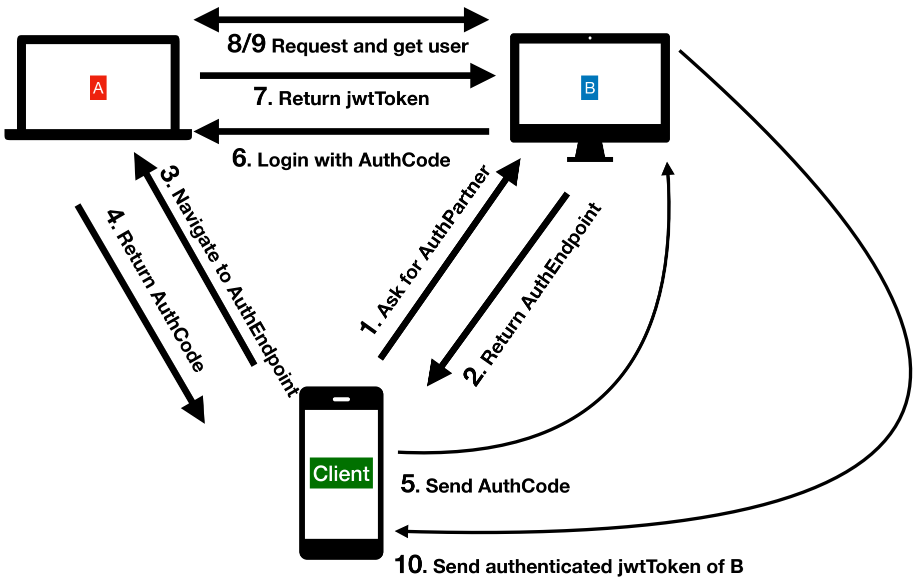
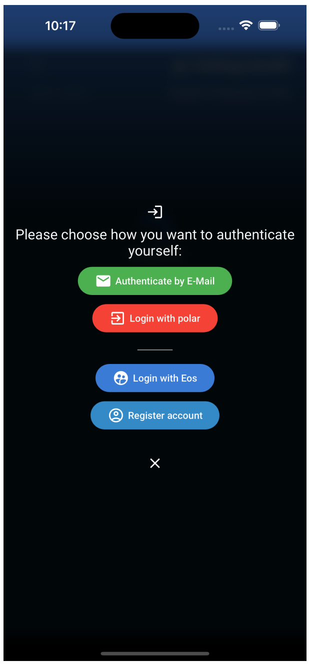


Figure 12: The client tries to authenticate at service B (e.g. Eos) with a user account from service A (e.g. Polar). This is called the **OAuth2 Authentication Code Flow**.



Please choose how you want to authenticate yourself:

Figure 13: Authenticate at Eos with a one-time-password sent to your e-mail, with AuthPartners (like Polar) or with a dedicated Eos Account.

Next to the name of the AuthPartner Eos needs four strings to enable third-party authentication partners:

- **authentication endpoint**: The endpoint where the user can enter their login credentials at service A. This url is sent to the client which shows this webpage in a webview. The webview returns an authentication code, which the client sends to the server.
- **token endpoint**: The endpoint where the service B (Eos) can exchange the authentication code for a jwt login token for service A.
- **user endpoint**: The endpoint where the server can request information about the user account on service A using the jwt token.
- **unique user variable**: Eos expects user information by service A in json format. Using the given unique variable name, it will query the json for the value of the user, save it in a user object of Eos and return an Eos jwt token which is authenticated for this Eos account.

4.3 Polar

4.3.1 Events Close to You

To suggest users events that are close to them, the Polar client sends the acquired current position to the server, if the user allows this. Let p be a position consisting of a latitude φ and a longitude λ. Let p_u be the position of the user. Let every event e_i (offline or hybrid) have such a position p_i as well. The distance between two positions p_1 and p_2 on a sphere can then be calculated by using the *Haversine* formula:

$$a = \sin^2(\frac{\Delta\varphi}{2}) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin^2(\frac{\Delta\lambda}{2}) \tag{1}$$

Haversine formular

The distance between p_1 and p_2 is then

$$d = R \cdot c \tag{2}$$

Distance

Where $c = 2 \cdot \mathrm{atan2}(\sqrt{a}, \sqrt{1-a})$ and R is the average radius of earth (in the desired output unit: meters or kilometers) [22]. φ and λ need to be radians. Polar performs this calculation directly on the relational database, first filtering for any event that is still ongoing (constraint c_1) and then applying the formula, which is translated by SQLalchemy to SQL code. This uses the SQL implementation of the functions *sqrt*, *sin*, *cos*, *radians* and *atan*2 (therefore needing a database that supports these functions). Using the built-in SQL functions is theoretically more efficient than loading all (ongoing) events from the database into the Polar server and performing the check in Python. If not requested differently by the user, the server then filters for every event where $d(p_u, p_i) \leq 50km$. This calculation is not accurate to the meter as it assumes that the earth is a uniform sphere. Still, it can be used well for this use case, as the distances are only needed approximately.

4.3.2 Motion Mood

Polar generates an overall mood m_g for each motion. It takes the mood $m_i \in [-2, 2]$ for each comment c_i and the amount of likes ϕ_i each comment got. Only top-level comments, where $m_i \neq 0$, are respected. The set of non-neutral comments is called C, and the length of C is c. We introduce a hyper-parameter $\Phi = 50$, that controls the influence of likes on the overall mood. $\phi_t = \sum_i^c \phi_i$ is the total amount of likes which all comments received. The formula is the following:

$$m_g = \frac{\sum_i^c (m_i + m_i \cdot \frac{\phi_i}{\Phi})}{c + \frac{\phi_t}{\Phi}} \tag{3}$$

Mood formular

This mood is calculated and returned when the client asks for it. The runtime for this calculation is obviously in $\Theta(c)$. See Chapter 6.3.1 for further comments on this.

4.3.3 Fuzzy Search

Polar uses Fuzzy Prefix Search [23] to enable users to search for relevant motions, events and comments. Fuzzy Prefix Search makes use of several ideas of computer science to enable users to get results even if they do mistakes in their search query. These ideas are Prefix Edit Distance, Q-gram indices and inverted lists.
An **inverted list** is a dictionary which keeps track which words are in which entry. For each word used in motions, events or comments, it keeps a sorted list with tuples. Each of these tuples contain the id of the entry o that contains this word and an occurrence factor s_o of how relevant this word is for this entry. If the motion M_1 and the event E_1 both contain the word *snow*, the following entry would be part of the inverted list:

$$\{snow : [(M_1, s_{M_1}), (E_1, s_{e_1})]\}$$

The occurrence factor is calculated using the **BM25 score** [24], which normalizes the effect of the frequency of the word in relation to both the length of the document of this entry as well as the average document length.

Once the inverted list i_n is built based on all titles and descriptions from every entry, a responsive and fast search is already possible. If the user searches for the query *snow*, we can do a simple lookup in our list and find all entries that contain the word *snow*. If we have sorted the list by the factor s_o, the most fitting entry will be returned first. The runtime of a lookup is in $O(n)$.

To allow users to enter a similar but wrong term (*snov* instead of *snow*) and still receive valid results, a feature known from famous search engines, we make use of the Prefix Edit Distance as well as Q-Gram indices.

Prefix Edit Distance is a variation of the Edit Distance. The Edit Distance $ed(x, y)$ is a metric which tells us how many REPLACE, INSERT or DELETE operations are necessary to get from word x to word y. The Prefix Edit Distance $ped(x, y)$ tells us how many operations are necessary to get from word x to word y' where y' is the word y capped at length of the word x. If x is a word with a mistake like *snov*, the (prefix) edit distance to words that are correct like *snow* is small. Therefore we are looking for words that have a $ped(x, y) \leq \delta$, with δ being small. The calculation of the PED for the search query and every word in the inverted list would be too time consuming. Therefore, we must make use of a trick.

This trick is called **Q-Gram Index**. A Q-Gram index $Q_q(x)$ is a multi-set of strings for a given string x dividing x into subsets of q length [23]. See this example:

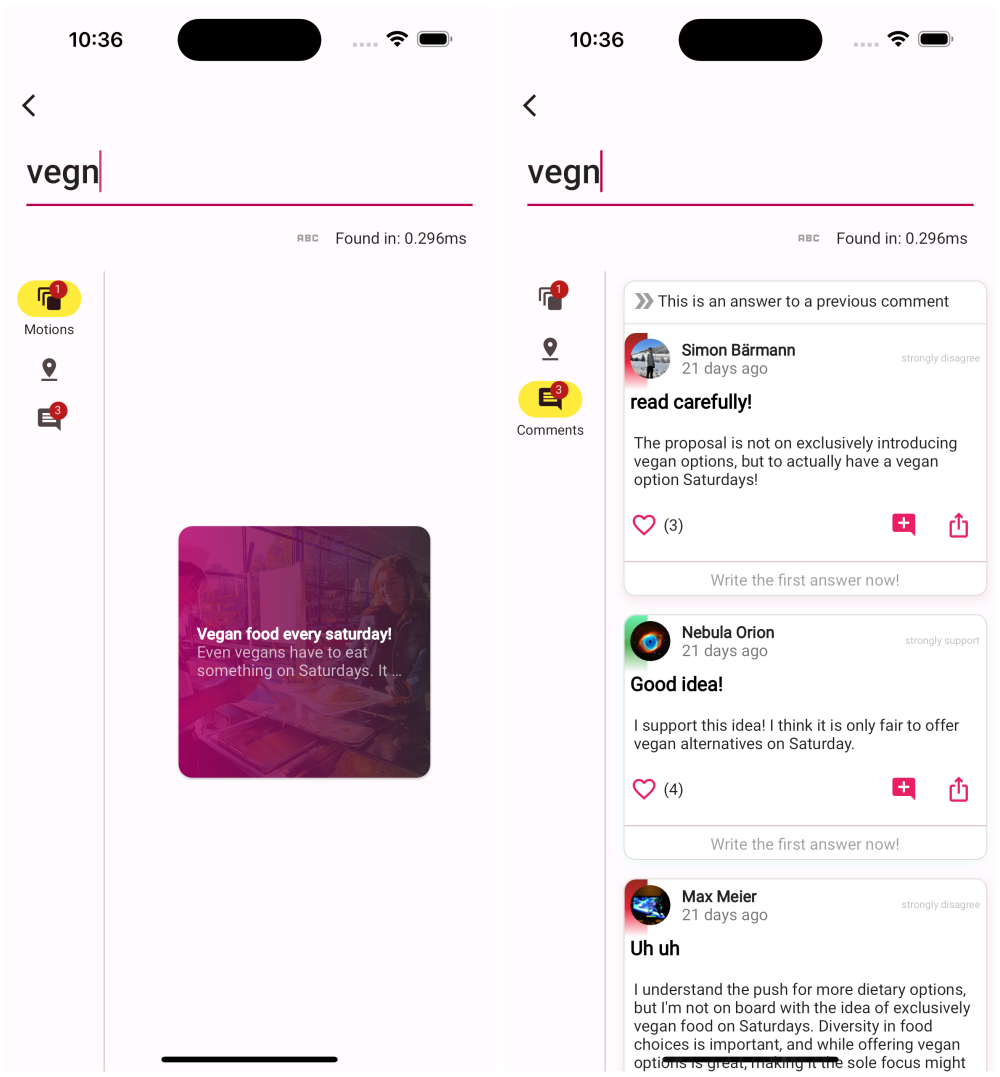
$$Q_2(polar) = \{po, ol, la, ar\}$$

Luckily, there is a corollary that tells us that $|Q_q(x) \cap Q_q(y)| \geq |Q_q(x)| - q \cdot \delta$ must hold true so that $ped(x, y)$ can be smaller equal δ. See [23] for a proof on this corollary. This means, that if two words have $|Q_q(x)| - q \cdot \delta$ Q-Gram indices in common, their $ped(x, y) \leq \delta$. Therefore, it is necessary to build another inverted list i_q.

This time, we are using each word that is part of i_n and build the Q-Gram indices for each. The keys of i_q are the indices of length q, the values a list of words that contain these Q-Gram indices. With i_q present, we must only calculate the Q-Gram index of x. Then we can lookup all Q-Gram indices of x and merge the corresponding word lists. We count the occurrence of words in the merged list. If one word y occurs $|Q_q(x)| - q \cdot \delta$ times, the $ped(x, y) \leq \delta$. An example entry of i_q would be: (in our actual implementation the words are replaced with their index in i_n)

$$\{ol : [polar, polestar, lol]\}$$

We return all entries in i_n for words that the user could mean. With this, we make sure that the user most likely finds what they are looking for, as can be seen in Figure 14. The runtime for Fuzzy Prefix Search contains multiple lookup and merging operations. It is therefore a more expensive search algorithm.



Vegan food every saturday!
Even vegans have to eat something on Saturdays. It ...

This is an answer to a previous comment

Simon Bärmann
21 days ago
strongly disagree
read carefully!
The proposal is not on exclusively introducing vegan options, but to actually have a vegan option Saturdays!
(3)
Write the first answer now!

Nebula Orion
21 days ago
strongly support
Good idea!
I support this idea! I think it is only fair to offer vegan alternatives on Saturday.
(4)
Write the first answer now!

Max Meier
21 days ago
strongly disagree
Uh uh
I understand the push for more dietary options, but I'm not on board with the idea of exclusively vegan food on Saturdays. Diversity in food choices is important, and while offering vegan options is great, making it the sole focus might

Figure 14: Searching for *vegn* still yields correct results!

5 Evaluation

To evaluate the results of this bachelor thesis, we conducted a user study. In this study, participants were asked on both their political engagements as well as their general thoughts on digital elections. They were introduced to Eos, guided to cast a vote in an election and got an explanation on the encryption protocol and how to verify their vote. Then they were asked on their opinion regarding the user interface, the intuitiveness of Eos as well as concrete questions on the election process and the encryption. The goal was to not only figure out how well-designed the app is, but also to see, whether or not Eos could convince participants to trust online elections more than they did before using the app. Following, they got introduced to Polar, guided through the different features and asked questions on each of them. In the end, they participated in a vote on a motion through the interlink of Eos and Polar. Finally, those who claimed they were active in political groups were asked how useful they think Polar would be for their (respective) political engagement.

The study was conducted in January 2024 using pre-release versions of Eos and Polar. The participants were further asked whether they study or work in the field of computer science. This information is used to figure out whether there is a correlation between deep knowledge of computer science and acceptance of online voting systems. The study was conducted in English and German and took between 40 and 60 minutes for each participant.

When participants were asked for a rating, they could choose a number between 1 and 5, with 1 being the lowest and 5 being the highest score. The study survey is attached at the appendix, see Chapter 8.1.

5.1 Scenario Setup

Eos was set up with three elections. Two of them were set to ongoing state, one already completed. The completed election was a mock election on the overall satisfaction of workers in a company with a variation of answer types. The results of this election were available and the votes cast by the voters decrypted.
The first ongoing election is an election on the adoption of the Universal Basic Income into the manifesto of an imaginary political party. This is highly influenced by the election conducted by the German party “Die Linke” (engl. the left) in 2023, which was done by mail.
The second ongoing election is an election on an imaginary student council. It is assumed that there are two chambers of which the first is composed by direct elections of candidates (a voter has five votes, which they can distribute among candidates), and the second chamber is composed proportionally, depending on how many votes are given to the factions (a voter has one vote, which they can give to one faction).
These three elections were picked as examples of what Eos could be used for. They were meant to showcase the various possible applications.
Polar was filled with example motions copied from the University of Freiburgs StuRa (student council). These motions were originally submitted to a general meeting, where all students of the university were invited to. The events are completely made up, but focus on a StuRa context. The motion and event chat rooms were filled with made up content. Wherever necessary, ChatGPT was asked for text placeholders to create an immersive experience.

5.2 About the Participants

Of the 8 participants, 2 identified as male, 6 identified as female and 0 identified as non-binary/other. All of them were in their twenties. 2 study or work in computer science. 4 are active in political groups or organizations, some in multiple. Of these politically active participants, all stated that their group already uses means to conduct their work digitally, where social media apps like Facebook or Instagram and chat messengers like WhatsApp or Signal were used tools mentioned most often. 1 stated that, in their opinion, these means are not enough. The participants were also asked whether they ever participated in an online election (4 out of 8), of which 0 said they actually understood how their vote was secured and how they could verify their vote. No one said that they really care about how their vote was secured. A comment here was that "it did not really matter, because it was only an election at university". 7 participants answered that they would generally trust online elections, 1 participant does not generally trust online elections. The reasons for this (dis-)trust vary. Those who trust online elections argued that they "trust the protection protocols", that they find online elections "very convenient", and stated that they simply would not care "if [their] vote was leaked". One participant argued that online elections in Germany would be safe simply because they would be held in Germany, while a participant from Hungary argued that they do not trust their in-paper-elections so online elections could only be better. The participant who does not trust online elections reasoned that they "feel like it is easy to manipulate".

5.3 User Interface and Experience

App	User Interface	User Experince
Eos	4.5 ±0.53	4.88 ±0.35
Polar	4.69 ±0.46	4.81 ±0.37

Table 3: User Interface and Experience scores. The participants were asked to rate the user interface and the overall user experience of both apps. The highest score participants could give was 5, the lowest 1. Both apps scored high and were well received.

For both Eos as well as Polar the participants were asked to rate the design of the user interface and the overall user experience. Both Eos as well as Polar received high scores for their user interface, as can be seen in Table 3.

The user interface of Eos was rated an average of 4.5 points out of 5 points. The participants praised the “professional” design, the “good overview”, the “pretty colours” and the overall “simple use” of Eos. Criticism was aimed at the “disproportional” design and some of the bigger empty spaces. Some elements of the user interface were also “too small”. However, no participant rated the user interface of Eos lower than 4 points.

The overall user experience of Eos was rated an average of 4.88 out of 5 points. The participants praised that the user experience of Eos is “pretty clear also to people who might be new to online elections”.

The user interface of Polar was rated an average of 4.69 out of 5 points. The participants praised the “clear, well designed” user interface, they found it “very elegant” and “easy to use”, while there was a suggestion to change the design of the bottom navigation bar. No participant rated the user interface of Polar lower than 4 points.

The user experience of Polar was rated an average of 4.81 out of 5 points. The participants acclaimed the “clear” experience on the app. One participant experienced a bug in the chat feature and criticised this. Overall, no participant rated the user experience of Polar lower than 4 points.

5.4 Eos

The main goal of the user study on Eos was to figure out whether the voting on Eos is intuitive, whether participants (think they) understand the Eos en-/ and decryption protocol and whether they would actually trust an election on Eos.

All participants praised the clean first impression Eos provides to new users. The intuitiveness of the voting on the Eos client was rated with 4.63 out of 5 points in average, meaning that the voting on the client is pretty intuitive for users ($\sigma = 0.52$). Out of all participants, 5 stated that they thought they understand the encryption

protocol. This was a self-assessment, the participants did not have to pass a test for this. All 8 believed that their vote is safe using this protocol. Some argued because the encryption protocol is very "transparent", others because of the implementation of "trustees". The computer scientists/workers both stated that they understood the encryption protocol as well as they believe that their vote is safe. They both agreed that they trust elections on Eos, as well as all other participants who also trust online elections in general. The participant who does not trust online elections in general also does not trust them on Eos. The participant stated that the reason behind this is that they do not study computer science and therefore do not really understand how encryption works. Later, the participant states that they would trust Eos with low-stake elections. Those who trust Eos argued that it seems "trustworthy", and that every voter can "check that the vote was [respected in the tally]".
Overall, Eos received good feedback in the user study. The user interface as well as the voting procedure were praised while it could not convince those, who do not trust online elections in general. The transparent design of the en- and decryption protocol left a good impression on the participants, with all of them extolling it. However, it can also be seen that the encryption and decryption protocol in the client should be explained more simply. Even if 5 participants state that they understand the procedure, probably only the computer scientists would pass a test on the procedure. Still, due to the responses received and the small sample size, no conclusion can be drawn as to whether knowledge of computer science has an influence on the acceptance of online voting, or specifically on Eos.

5.5 Polar

Four features of Polar (events, chats, motions and the connection between Polar and Eos) were shown to the participants. For each feature the participants were presented with the same questions. How they would rate the usefulness of this feature for political groups, how necessary they believe this feature is for Polar's goal (to centralize all (digital) workflows of political groups) and what further questions or comments they have for this feature, as can be seen in Table 4. The motion and the connection feature both scored a perfect 5 out of 5 points on their usefulness on average. The motion feature received high praise by the participants, it was called "very useful". The connection feature also received many compliments, especially that users can use their Polar account and are automatically signed in to the Eos client. The connection feature also scored a 5 out of 5 on average regarding how necessary it is. The motion feature got 4.88 out of 5 points on average for this question.

The chat feature was ranked lowest on usefulness with 4.5 out of 5 on average, while it has received the same necessity ranking as the motion feature, with 4.88 out of 5 points on average. Participants questioned whether users really would use the Polar chat feature and not other chat apps they already frequent often. However, to achieve Polar's goal, the feature was still deemed necessary.

The event feature scored 4.75 out of 5 points on average for usefulness, while it scored 4.81 out of 5 points on average regarding how necessary it is for Polar's goal. The event feature therefore got the lowest score on this question. Participants praised the design, but raised concerns on the missing support of "recurring events".

Feature	Usefulness	How necessary?
Events	4.75 ±0.46	4.81 ±0.53
Chats	4.5 ±0.76	4.88 ±0.35
Motions	5 ±0	4.88 ±0.35
Connection	5 ±0	5 ±0

Table 4: Feature ranking. The participants were asked for each feature, how useful they think the feature is for political groups and how necessary the feature is to achieve Polar's goal of centralizing (digital) workflows of political groups. The highest score participants could give was 5, the lowest 1.

We can see that all of these features were overall ranked as useful for political groups, their implementation seen by the participants as necessary to achieve the goal Polar is tasked with. **All politically-active participants stated that they could see their political group or organization using Polar**. Some argued that political groups consisting of young people could especially benefit from Polar. For example, the “Jusos” or the “Linksjugend [’solid]” were mentioned, both left-wing youth organizations. The student council setup of the app was applauded by the participants, and they stated that the app could be used for the student council. A member of a small political group stated that they cannot see Polar as an replacement of their offline workflow but definitely as an “addition”. Overall, all participants rated the usefulness of Polar for political groups and organizations with 4.88 out of 5 points on average ($\sigma = 0.35$).

Both apps received overwhelming positive responses, while some limitations apply to these results. Read more about them in Chapter 6.4.

6 Limitations

Eos and Polar both come with limitations and a long list of possible features that could be implemented. Eos, especially if it is used for a high-stake election, could be attacked by malicious entities. First, we will discuss these attack vectors.

6.1 Attacking Eos

6.1.1 Corrupt Server

The worst case scenario is a corrupted Eos server. This is rather easy for an attacker with admin rights to the computer where the Eos server runs on, as they can easily change the code of Eos. Attackers could leak all stored sensitive personal information. Further, they could simply implement a map between new cast votes and user objects, hereby removing the secrecy of the election. This is not possible for elections that have already been completed, but they can leak which voter has voted in general. Still, for elections with $n > 1$ where n is the amount of trustees, an attacker could only decrypt the votes before the election is over if all trustees are corrupt as well. This kind of attack is easily visible for security-conscious hosts of Eos but rather hard to see for users of Eos.
A corrupt Eos server could also copy votes in the CastVote table and thus distort the election result. A corrupt Eos server could also let people vote twice or allow the casting of votes by non-eligble voters. There are currently no measures against this

behavior. A corrupt Eos server set up by a malicious admin would therefore be the worst-case scenario and could lead to falsified results. Following proposed features could mitigate this problem to the extent that the corrupt server would have to make a high effort to bypass them.

1. **Voters list**: Implement a feature where the admin of the election can see which eligible voter has already voted. The server could fool the admin that malicious votes by non-eligible voters belong to eligible voters. But the server could only spoof a fixed amount of votes. Currently Eos does not release the list of users who have voted on the election to neither the admin of the election nor to anyone else. This list is only available by directly accessing the database. However, if Eos were to publish this list of voters, it would be possible to draw conclusions about voting behavior, especially in elections with few participants.

2. **Shared cast votes list**: Changing the design of Eos so that the user submits their vote to every trustee (therefore having n copies of the castvotes list), could prevent the Eos server from spoofing votes. First, trustees could prevent multiple votes from the same IP (or mark them as suspicious) as well as they could demand a proof from the client submitting the vote that they are human using Googles ReCAPTCHA for example. This would demand a change in the eligibility check system, which either needs to happen on every trustee as well or with a validation code which the Eos server delivers to the client with which the client then authenticates their eligibility to the trustees. The latter being the more favorable design choice, as trustees obviously do not have access to the user system of Eos.

3. **Blockchain-like signing of votes**: Every client could generate a RSA key pair when they cast a vote to the server. The client signs the encrypted vote with the generated private key and submits the signed encrypted vote as well as the public key to the server. The client saves the private key. (This would also allow to revoke or change votes once they are cast). The client could also

download the hashes of all previous cast votes and then hash and sign this list as well (with the private key), adding this signed hash to the encrypted vote. If the server then swaps or drops previous votes, the hash would not match anymore (this obviously would make the revoking feature obsolete again). This feature has some downsides as it would basically stop users from casting at the same time. Also, the size of encrypted votes would grow as would the runtime needed to cast votes, especially the generation of secure RSA key pairs takes computation power. Using one general key pair per user would obviously not work, as it would be equal to signing a ballot with your own signature and make all secrecy measures obsolete.

4. **Signed server proof**: The server should sign the hash of the vote the user has cast to the server as a proof with the private key of the Eos trustee. The client should receive this proof and save it. By saving it, the client can then prove if the decrypted vote differs from the actual vote that this client cast.

Even without the proposed features, a corrupt server could not change a submitted vote without changing the hash of the vote, vigilant voters or experts observing the elections would notice this. Further, a corrupt server could not simply edit the calculated result object without also tampering the decryption results. Either the manual counting (by experts) of the results would not be correct or vigilant voters would see that their cast vote does not equal the decryption result. Tampering with trustee results (not actually decrypting the votes but just publishing a made up result) would therefore also not work. If only one vigilant voter remembers what they voted for, they could call for the nullification of the election.
We can see that adding further security measures would complicate the casting process as well as make it more (resource) expensive. The current protocol was therefore chosen to enable the simplest comprehensibility and verifiability on the one hand, but also to hold true the following axiom:
The protocol shall be designed so that it is too arduous to manipulate low-stake

elections and it is easy to prove that manipulation has taken place in high-stake elections.
One could argue that this does not completely hold true for corrupt servers. But it will be visible that the current design is sufficient to fulfil this axiom against other attack vectors. In addition, the implementation of the proposed functions would increase the protection even against corrupt servers. Still, users should generally trust the host of the Eos server and not vote on hosts that look suspicious.

6.1.2 Corrupt Clients

Corrupt Eos clients could either arise from corrupt user devices or from using a third-party Eos client. Users could either willingly use a third-party client or be fooled for example by phishing mails into thinking they are using the official client. Corrupt clients can only ever tamper with the vote of one user, they can not submit votes of non-eligible voters nor can they change already submitted votes. Corrupt clients could steal login credentials like username and password as well as steal the login token used to authenticate users while they are logged in. The official Eos client allows the user to verify their encrypted vote. If something looks fishy, they should not cast their vote. If their encrypted vote hash changes from before to after the cast process, they know their vote was tampered with. If the hash stays the same and the server reports the same one, if the user furthermore computes the hash for themselves (the official client and server easily allow to do this) from the cast votes list, and this computed hash is also the same, they can be pretty sure that their vote was not tampered with. Once the decryption result is released they can verify one more time that their vote is correct. The attack vector of corrupt clients therefore is small and only affects non-vigilant voters. Still, likely this would be the most used attack vector on elections as it is the most easy one. A possible feature to eradicate this attack vector would be:

5. **Allowed clients**: Only accept clients to submit and cast votes, if they were

previously allowed. This would prevent phishing attacks as well as make sure that the cast votes are encrypted correctly. Allowed clients could either be implemented using standards like the OAuth2CredentialsFlow or by limiting the access to the server from only the ip of the client web app (this would render invalid all native Eos clients).

6.1.3 Corrupt Trustees

Corrupt third-party trustees could reveal their private key to malicious attackers. This would only result in the decryption of one of the layers of each encrypted ballot. The actual vote of the user can not be decrypted without having access to all private keys of all trustees n. A corrupt trustee could respond with spoofed decryption result where decryption results are imagined. This is possible, but would be visible in the end as the released decryption result of the vote would not equal what the voter has voted. Vigilant voters would therefore realize that their vote was tampered with.
Corrupt trustees could stop an election from ever being tallied if they simply refuse to decrypt their encryption layer. The election would have to be started again without this trustee.

6.1.4 Flooding Eos

Maybe an attacker does not actually want to spoof votes, but rather just disturb the election process. Eos is currently not rate-limited, so any client could flood the server with requests. Most queries are not really resource-intensive, simple queries to the database often only take a few milliseconds. However, uploading of images in particular could be exploited to bring the server to a standstill with a rather small amount of requests, as the server currently calculates a so-called blurhash for uploaded images, which turns the image into a simple and short sequence of bytes that can be displayed while the user downloads the actual image, as can be seen in Figure 15. This calculation can take a few seconds, a perfect attack vector to paralyze the server (and thus disrupt elections). Following features/design changes could mitigate the attack vector:

6. **Blurhash client-side**: The client could calculate the blurhash and simply provide it to the server after uploading the image. The server has to accept the blurhash of the user and will not check it. Image sizes are already limited by the server-application hosting the Eos server (like uvicorn or apache), reducing the maximal image size further mitigates the problem.

7. **rate limit queries**: Introducing rate limits could mitigate the problem, as any client overexceeding rate limits would be banned from sending new requests for a defined duration. Working together with DDoS protection services from commercial providers could secure Eos servers more from attacks like these. Arguably, that the latter is only necessary for high-stake elections.

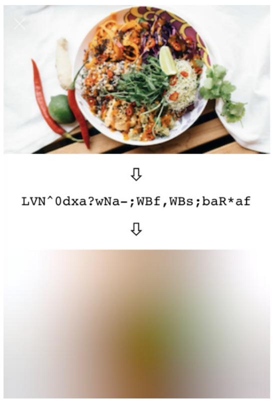


Figure 15: The blurhash, an object a few bytes in size that can be used as a placeholder, is calculated from an image [25].

The positive aspect of this kind of attack vector is that it is very easy to recognize. An election process attacked under this vector could be interrupted until the attack is over.

To disturb the decryption process, an attacker could use an open election (where no eligibility constraints are set) and submit a vast amount of invalid votes. These votes would not change the result but disturb the decryption process. The Eos server does not currently check whether the encrypted vote is correctly encrypted, it accepts any map object. A simple millions-of-bytes-big map of gibberish would also be accepted. The server could therefore be flooded with invalid votes which are expensive size-wise. To prevent this, refer to former proposals like Feature 5, the implementation of Google ReCAPTCHA and the following feature:

8. **Verify votes during submission**: The server should make sure the votes do match the expected size, as votes with q amount of questions and n amount of layers are usually in a definable size frame. Votes much smaller or bigger than this size could be simply rejected.

6.1.5 Quantum-safe Encryption

Neither RSA nor AES are quantum-safe [26]. This means that corresponding progress in quantum technology will at least enable those who have access to it to crack RSA and AES. However, this only applies to currently ongoing elections, as cast votes are decrypted nevertheless after the election is over. Currently and for the foreseeable future, this attack vector is limited to only a small amount of highly powerful actors and definitely neglectable for low-stake elections. If RSA and AES are cracked, a lot of software would be broken [26]. Still, there are so-called post-quantum encryption protocols. These are much more complicated than RSA and AES but would allow votes to stay safe even in a post-quantum world. One of these proposed post-quantum encryption protocols is CRYSTALS-Kyber [27], which also works with public and private keys.

9. **Quantum-safe encryption**: Changing the Eos En- and Decryption Protocol to quantum-safe encryptions would mitigate attacks on RSA and AES as both are not used anymore.

These nine proposed changes or additions to the design of Eos would mitigate attack vectors on Eos further. But there are also some limitations which are not directly abusable by attackers.

6.2 Eos Limitations

6.2.1 Ballot Size and IDs

The size of encrypted ballots grows **exponentially** with the amount of trustees n. We saw in Table 1 that a ballot with $n = 8$ is around 15 megabytes big each. Using a lot of trustees to secure the election is therefore a highly expensive task. But, arguably, it is rare that there is a need for this many trustees. Eos should not be used for high-stakes elections in general, as no online election service should be. $n = 3$ trustees is completely sufficient for low-stake elections, assuming that the two third-party trustees are genuinely different. Encrypted ballots for $n = 3$ are around 26 kilobytes big. This means for every encrypted ballot with $n = 8$ trustees, around 600 ballots with $n = 3$ could be stored.
CastVote objects should not use a numeric id as their unique identifier and the timestamp they were added should be removed from the database before the votes are being counted. It should not be possible to reconstruct a chronological sequence of votes.

6.2.2 Missing Features

Some features of Eos were not implemented.

- Users creating an account through an authentication partner (AuthPartner) like Polar can not convert their account to a full Eos account. AuthPartner accounts usually do not have a username nor an email. Converting their account to a full Eos account simply means adding an username and an email. This is currently not possible.

- Clients are not saving the users voter alias nor the hash they showed. Users can create a screenshot. Without a screenshot they can not be sure which vote is actually theirs, if they have forgotten their alias.
- It is currently not possible to upload a list of eligible voters in a file. Every voter has to be added through the user interface and be searched individually.
- Trustees added to an election can only be removed through the database.
- Elections can only be flagged as featured through the database.
- Every user can create elections. There is no user role system.
- Currently the Eos client saves bookmarks locally and not on the server. This could be argued is strange behaviour because bookmarks are then available to all users using this end-device. But one could argue that Eos should be only used on your own device to begin with.
- Users can not currently reset their password if they have forgotten it. They can not delete their account.
- Even though the flag exists in the database, write in answers and randomized answer orders are currently not implemented.

6.2.3 Maximum Amount of Voters per Election

The amount of voters is limited by the possible unique voter alias. This is dependent on the amount names of animals Eos is provided with. Currently those are 235 animal names. Eos generates a random number with the length $i = 5$ in $[10000, 99999]$ and adds this number to the animal name. If a voter alias for this combination on this election already exists, the voter has to **repeat** the casting. The amount of voters is therefore limited by:

$$235 \cdot 89999 = 21.149.765$$

voters. Arguably, this limit is not a big problem, as one can easily split elections if they expect this amount of voters into multiple separated elections. (For comparison: German federal elections have around 400.000 eligible voters each constituency, the biggest German union would ask around 2.1 million voters).
Eos should be further adapted to not return an error if it generates a voter alias which already exists but should generate a voter alias again until it finds one that is valid. Additionally, it can be easily extended by increasing the amount of digits on the random number.

6.2.4 Limit and Offset

The Eos client currently hardcodes the length of the requested answer of a query, such as the request to get all featured elections, to the server. This length l is currently 100. If there are more than l entries in the database they are not sent to the client, as the client did not request them. The Polar client does not have this issue, as it features a 'load more' button. This implementation needs to be added to Eos as well.

6.2.5 Timezone Mixture

The election start and end times depend on the timezone the server uses. But the client currently respects only the timezone of the device. So an election could be displayed as already over, even if the election has not ended on the server yet. This can be changed by always translating all time objects to a specified timezone and then only respecting this timezone.
Ballots are still only cast if they are submitted during the start time and end time in

regards to the servers timezone. A client that displays that the election is still going on will not lead to a cast vote after the election is actually over. Obviously the server will reject votes cast before the election has started as well.

6.3 Polar Limitations

Polar has limitations as well. There are no special attack vectors on Polar, because Polar does not perform elections or similar things. Attackers could try to get sensitive personal information of users, but Polar has measures to prevent this. Furthermore there is only a limited amount of API calls that are available for unauthenticated users.

6.3.1 Motion Limitations

While the motions feature of Polar can be used for a variety of imaginable use-cases and while fine-tuning possibilities are given, the automatic motion state switch algorithm of the server does not respect all use cases. It can be necessary that the motion commission switches a state manually and deletes previously set timestamps. The motion commission also needs to check in the end whether the election on the final adoption of the motion was successful or not and then manually change the state to adopted or rejected. Polar does not display the results of the election, users have to navigate to Eos to see the results.
The mood score of motions is made of the mood of top-level comments the motion got in the deliberation phase. It is factored with the likes a comment got. This score is an indicator on the moods of the received comments. It is not an indicator on how the election on this motion will conclude. Motion moods could be calculated directly on the database, this would make the calculation more efficient.
Comments on motions still can receive likes altering the mood score of the motion even when the motion has shifted post the deliberation phase. This could be atypical

in the understanding of the phase model, as new comments are not allowed outside of this phase.

6.3.2 Event Limitations

Events have limitations as there are no event series. This means weekly events need to be created every week. Also, events are only available to logged-in users while one could argue that it should be possible for unauthenticated users to have access to events specially marked as public as well.
Events marked as online without a physical address are not as featured as offline events in the Polar client. The client only sends the request to get events the server once it has acquired the location of the user or when a timeout has occurred. This should be changed so that the user does not have to wait for the events till the sometimes slow location acquisition is done.

6.3.3 Chat Limitations

The chat feature of Polar is practically outsourced to Matrix. The Polar server only does limited manipulation of the Matrix server as in it only creates new public chat rooms. The client is limited in supporting only the basic messaging features. Also, the client does not currently support encrypted chats as well as it does not support push notifications. However, the usage of Matrix still allows the user to use any third-party chat service. Implementing encrypted chat rooms into the Polar client is not very complex as the Matrix package which handles the communication to Matrix does the most work on this. Still, implementing encrypted rooms was out of the scope of this work. The chat feature should be more seen as a proof-of-concept for now. It is possible to use a well established communication protocol (Matrix) and support chat between users. With more effort being put in, features like sending images or voice messages could easily be supported.

Further, the Polar client currently does not support searching chat rooms for messages.

6.3.4 Missing Features

Future work of Polar could contain the following missing features:

- Users cannot search for other users. Direct messages can only be established if the other user can be found through other chats, as the author of an motion, event or comment or in the responses / endorser lists.

- There is currently no admin page for comments. The moderation capabilities of comments is limited. Same holds true for amendments.

- There are currently no SSO scopes. The returned token in the SSO authentication process allows the third-party service to access all API calls. This should be changed: third-party services should be registered first by the Polar admin, defining exactly what they are allowed to access.

- Polar users can neither request a new password in case they forgot theirs, nor can they (or an admin) delete their account yet.

- Amendments should have comments and likes as well.

- While Polar acquires a lot of information on what users like and endorse, this is not used to tailor personalized recommendations in motion or event feeds.

- Polar should send notifications on when another user significantly interacts with your motion, event or comment. Also, it should send notifications on starting elections. An easy implementation would work with the already existing email system. Another option is to implement notifications like on famous social media platforms as well as push notifications.

- Allowing admins to archive motions and motion groups could really benefit the user experience on the app from feeling too bloated. Archived content could appear only in a specific subpage of the app or only be displayed in the search.

- Exporting motions to import them into other services like 'OpenSlides' [15] could enable a smooth transition between Polar and existing services. Polar should also support an import feature to accept motions and events from third-party services.

6.4 User Study

The user study was performed on a small, homogeneous group. While the study provides an initial overview of satisfaction with the apps and the design, no universally valid conclusion can be drawn from it. For one, the study design should have made it clear that Eos is an app for "low-stake" elections. In addition, the understanding of the encryption process should not only have been queried via a self-assessment. The question of whether participants believe that their vote is safe on Eos is biased to a certain extent, as participants trusted the study conductor overall. The study should have been done with participants that do not know the conductor, and with conductors, that are not the developer of the app. Due to the homogeneity of the participants in terms of their age, the study cannot make any statement about how appealing Polar is to politically active people who are not young. Thus, the user interface and user experience could also be rated differently by people who are not digital natives than it was rated by the young study participants.

7 Conclusion

In this thesis, we introduced a proposal to provide political groups and organizations with a suitable all-in-one software solution that allows them to operate their typical workflows in the digital space. This software solution consists of two applications: Eos (Election Online Service) and Polar (Politische Arbeit). Eos supports verifiable online elections with multiple trustees, various question types and very fast tallies. Polar supports collecting, processing and voting on motions, sharing and discussing events and communicating in (private) chat rooms. It works as a single-sign-on provider, so that other services can use Polar as their identity provider.

We dived into particularly interesting algorithms, like the Eos En- and Decryption protocol, which aims to enable both secure and easy-to-verify voting using a split-up system to en- and decrypt votes, where all parties of this systems would have to be corrupt in order for the secrecy of the election to be in danger. We also analyzed Polar's closest events feature, which calculates the distance between user and events elegantly directly on the relational database. Polar's fuzzy prefix search allows a fast lookup of motions, events and comments which is forgiving in the event of minor input errors (like a missing or wrong letter).

The conducted user study resulted in overwhelmingly positive responses for both Eos as well as Polar. Not only were both apps and the features of each app rated highly by the users, all of those who were politically active stated that they could see the use of Polar in their political group or organization. The user study also indicated that while the aim of the Eos En- and Decryption protocol was to be easily understandable, the client should be expanded to include further simple explanations.

In the limitations, we tried to shed light on the attack vectors that Eos offers and how elections can be manipulated and disrupted. We came to the conclusion that Eos suffers from the typical symptoms of online voting systems, while it easily allows users to make sure that their vote was cast, respected and correctly valued. This distinguishes Eos from other online election systems. For low-stake elections, Eos can therefore be a good choice despite the existing attack vectors. Furthermore, we introduced feature proposals which would implement further layers of security and further abilities like the one to revoke cast votes (Feature 3).
We analyzed limitations of both Polar and Eos and came to a list of additional features that future work on both apps should implement. Most of these features are of rather low-effort, and only did not make it into this version due to the severe time constraints of the work.
While future work could further enrich both apps, Eos and Polar are ready to use and can be deployed today to provide political groups and organizations (and beyond) with a centralized solution for digital participation.

Bibliography

[1] S. t. Ramírez, "Fastapi," 2024. `https://fastapi.tiangolo.com/` [Accessed on 07. February 2024].

[2] M. Bayer, "The python sql toolkit and object relational mapper," 2024. `https://www.sqlalchemy.org/` [Accessed on 07. February 2024].

[3] M. Makai, "Object-relational mappers (orms)," 2022. `https://www.fullstackpython.com/object-relational-mappers-orms.html` [Accessed on 07. February 2024].

[4] Google, "Flutter: Build apps for any screen," 2024. `https://flutter.dev/` [Accessed on 07. February 2024].

[5] B. Adida, "Helios: web-based open-audit voting," in *Proceedings of the 17th Conference on Security Symposium*, SS'08, p. 335–348, USENIX Association, 2008.

[6] M. Ogburn, C. Turner, and P. Dahal, "Homomorphic encryption," *Procedia Computer Science*, vol. 20, pp. 502–509, 2013. Complex Adaptive Systems.

[7] NemoContra, "Security," 2024. `https://nemovote.com/security/` [Accessed on 07. February 2024].

[8] ElectionBuddy, "Services," 2024. `https://electionbuddy.com/services/` [Accessed on 07. February 2024].

[9] electionrunner, “electionrunner,” 2024. `https://electionrunner.com/support/kb/results/election-results` [Accessed on 07. February 2024].

[10] S. Bell, J. Benaloh, M. D. Byrne, D. Debeauvoir, B. Eakin, P. Kortum, N. McBurnett, O. Pereira, P. B. Stark, D. S. Wallach, G. Fisher, J. Montoya, M. Parker, and M. Winn, “STAR-Vote: A secure, transparent, auditable, and reliable voting system,” in *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13)*, USENIX Association, Aug. 2013.

[11] C. Cassidy, “Explainer: Voting systems reliable, despite conspiracies,” *Associated Press*, 2022. `https://apnews.com/article/2022-midterm-elections-technology-voting-donald-trump-campaigns-46c9cf208687636b8eaa1864c35ab300` [Accessed on 07. February 2024].

[12] C. C. Kling, J. Kunegis, H. Hartmann, M. Strohmaier, and S. Staab, “Voting behaviour and power in online democracy: A study of liquidfeedback in germany’s pirate party,” *CoRR*, vol. abs/1503.07723, 2015.

[13] L. D. e.V., “Adhocracy+,” 2024. `https://adhocracy.plus/` [Accessed on 07. February 2024].

[14] Berlin, “Berlin mitgestalten: Ihre meinung ist uns wichtig!,” 2024. `https://mein.berlin.de` [Accessed on 07. February 2024].

[15] OpenSlides, “Funktionen in openslides,” 2024. `https://openslides.com/de#features` [Accessed on 07. February 2024].

[16] Matrix-Org., “Matrix specification,” 2024. `https://spec.matrix.org/latest/` [Accessed on 07. February 2024].

[17] Synapse, “Synapse,” 2024. `https://element-hq.github.io/synapse/latest/` [Accessed on 07. February 2024].

[18] Element, “Secure collaboration and messaging,” 2024. `https://element.io/` [Accessed on 07. February 2024].

[19] E. Milanov, "The rsa algorithm," 2009. https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf [Accessed on 07. February 2024].

[20] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, "Advanced encryption standard (aes)," November 2001.

[21] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000. https://ics.uci.edu/~fielding/pubs/dissertation/top.htm [Accessed on 07. February 2024].

[22] C. Veness, "Movable type scripts: Calculate distance and bearing between two latitude/longitude points using haversine formula in javascript," 2020. https://www.movable-type.co.uk/scripts/latlong.html [Accessed on 07. February 2024].

[23] H. Bast, "Dbis, lecture 7: Fuzzy search, ped, q-gram index @ 05.12.2023," 2023.

[24] H. Bast, "Dbis, lecture 2: Ranking and evaluation @ 24.10.2023," 2023.

[25] Image by Dag Ågren, released under MIT license. https://github.com/woltapp/blurhash/blob/master/Media/HowItWorks1.jpg [Accessed on 07. February 2024].

[26] K. Houston-Edwards, "Quantum-proof secret," *Scientific American Magazine Vol. 330 No. 2*, p. 36, Februrary 2024.

[27] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals – kyber: a cca-secure module-lattice-based kem." Cryptology ePrint Archive, Paper 2017/634, 2017. https://eprint.iacr.org/2017/634 [Accessed on 07. February 2024].

8 Appendix

8.1 User Study Survey

See the survey on the next page. Explanation texts were read to the participant. Action steps were performed together with the participant, as in they were either given the instruction to do something or the conductor performed the action.

Explanation:

Welcome to the user study on Polar and Eos. Thank you for agreeing to participate in my user study and to help me evaluate my results of my bachelor thesis. The results of my thesis try to centralize the work of political groups and organizations in the digital space meaning that there should be one software suite for the tasks that such groups generally have to perform.
First, I will present you Eos, an app to conduct safe online elections. Eos stands for “Election online service” and is a standalone app. Users and tools can create elections where predefined eligible voters can vote on. The vote is encrypted using the Eos En-/ and Decryption Protocol. The goal of this protocol is to make sure that your vote is safe and that you can verify that the released result in the end contains your vote.
Then I will present you Polar, which stands for the German word “Politische Arbeit”. Polar is meant as the central hub for political groups or organizations in the digital space. It supports a motion feature, where groups can collect, discuss, and vote on the adoption of motions. It supports an Event feature where users can share and comment on interesting events and it supports a Chat feature where users of Polar can chat with each other. To vote on Motions, Polar uses Eos to ensure that the voting is secure. Polar can therefore be used as a Single-Sign-On-Provider which means that users can login on other services (like Eos, or NextCloud) with their Polar account.

But let us start with some basic questions.

User-Study on Eos and Polar

P is the Participant. 1 means low, 5 means high. &/: and or. /: exclusive or. ?: voluntary.

A0	**P** identifier	
A1	What **gender** do you identify with? (f/m/nb/?)	
A1.1	What is the first number of your **age**? (1/2/3/4/5/6/7/8/9/?)	
A2	Are you a **student** of or do you **work in** the field of **computer science**? (y/n)	
A3	Are you **active** in a **political group** or **organization**? (y/n)	
A3.1	If so, **where** are you active? (text?)	
A3.2	Does your political group or organization **use any digital means** to communicate or work in the digital field? (y/n/?)	
A3.3	If so, **what means** are used? (text?)	
A3.4	Do you think that these means are **enough**? (y/n/? &/ text)	
A4	Did you ever conduct an or participated in an **online election**? (y/n)	
A4.1	Did you understand how and in what way your vote **was secured ///** and	

	how you could **verify** the election result? (y/n/?)	
A4.2	Did you care? (y/n/?)	
A5	Would you **trust an online election** in **general**? (y/n)	
A5.1	**Why** / Why not? (text)	

Eos

1. Open Eos and display front page, which is filled with three featured Elections (two ongoing, one completed).
2. Let **P** click trough the app.
3. Ask E1.
4. Cast a vote with **P**.
    a. Open one of the ongoing Elections.

Show trustees:
This election as you can see has two trustees. One trustee is the Eos server. The other trustee is a third-party trustee. What do trustees do? Each trustee has a public key, with which you vote is encrypted and a secret private key, it does not publish, with which it can decrypt your vote. This means, that only if all trustees agree to decrypt, your vote will be decrypted. This makes sure, that your vote is not decrypted before the election is over and that no one can see how you have voted.

Show votes:
Here we can see that some votes were already cast. The name you see is a random alias this voter got. Each time you vote, you get a new alias only for this election. Below that you can see the hash of the encrypted vote. If we click on it, you are given tools to actually build this hash for yourself, to make sure, no one tampered with your vote. Once the election is over, the decryption result is released here. So if you remember your alias, you can make sure, that the result of the decryption equals what you have voted for. This way, you can make sure, that your vote was actually respected in the result.

    b. Open ballot page.
    c. Let **P** answer the questions.
    d. Show how the verification work.

Verification:
Here you can see how the verification of the vote works. You have voted this (blue box). These are the internal ids of the answers. If we press on the first encryption layer, we can see that your vote was encrypted with a generated secret and the public key of the first

trustee (Eos). If you have the tools, you can try this for yourself, to make sure that the encryption is correct.
If we open the second layer, we can see that this first layer was encrypted with the public key of the second trustee. We can also see the generated hash of this encryption. This is your encrypted vote. Because your secrets were exposed now, we have to re-encrypt your vote again, with new secrets, but you can verify this again until you trust the encryption protocol.

e. Let **P** cast the ballot and display the authentication process.

As you can see, here is the alias you got assigned and the hash. With this, we can go back.

f. Show the cast vote in the votes list.

Here is your vote in the voters list.

5. Ask E2, E3.
6. Let **P** cast a vote again for the same election.
7. Ask E4, E5.
8. Let **P** click through the app till they are done exploring.
9. Ask remaining E questions.

E1	What is your **first impression** of Eos? (text)	
E2	How **intuitive** did you think the **cast process** was? (1-5 &/ text)	
E3	Are there any **questions** remaining? (text?)	
E4	Are you **understanding the encryption protocol**? (y/n)	
E5	Do you think your **votes are safe**? (y/n)	

E5.1	If not, why not? If yes, why? (text?)	
E6	Are there any **questions** remaining? (text?)	
E7	How would you **rate the user interface**? (1-5)	
E7.1	**Why** would you rate it this way? (text?)	
E8	How would you **rate** the **overall user experience**? (1-5 &/ text?)	
E9	(→ see A5) Would **you trust** an online election on Eos (now)? (y/n)	
E9.1	**Why** (not)? (text?)	

E10	Any **further comments** on Eos? (text?)	

Polar

1. Open Polar, which contains multiple motions, events and is connected to Matrix.
2. Show **P** the front page.
3. Start with Events. Go to Events, where multiple events are located.
4. Show an Event page.
5. Ask P1, P2 and P3.
6. Press on the user that created the event and start a chat room with them.
7. Then go back and enter the group chat of the event.
8. Write some messages.
9. Ask P4, P5, and P6.
10. Go to Motions. Open Motion **P** is most interested in. Show the user interface, explain the motions current process and where in the motion process its situated.

Motion Process:
A motion can be in different phases. They start as drafts, then they reach the quorum phase, where they are looking for supporters. If they find enough endorsements, they reach the deliberation phase where they can both collect comments as well as amendments. Next they are in the negotiation phase, where users can vote on the amendments. And then it reaches the election phase, where users vote on the general adoption of motions. Between these phases are validation phases, where users with special rights (the motion council) perform organizational tasks: checking whether the motion is allowed, summarizing amendments or adapting accepted amendments into the proposal. Motions do not have to follow this order, the motion council can freely shift them through these phases.

11. Let **P** explore the motions. Show comments and Amendments.
12. Ask P7, P8 and P9.
13. Open Motion with currently ongoing election. Go to the Election-button and navigate to election. Let **P** vote.
14. Ask P10, P11 and P12.
15. Let **P** roam through Polar till they are done.
16. Ask remaining questions.

P1	How **useful** do you find the **Event** feature (for political groups)? (1-5&/text?)	

P2	How **necessary** do you think this **feature** is in the context of Polars goal*? (1-5) * Centralizing digital workflows of political groups.	
P3	What are **further comments/questions** to **Events**. (text?)	
P4	How **useful** do you find the **chat** features of Polar (for pol. Groups)? (1-5 &/ text)	
P5	How **necessary** do you think the **chat features** are in the context of Polars goal? (1-5)	
P6	What are **further comments/questions** to chats? (text?)	
P7	How **useful** do you find the **motions feature** of Polar (for pol. Groups)? (1-5 &/ text)	
P8	How **necessary** do you think the **motions feature** is in the context of Polars goal? (1-5)	
P9	What are your **further comments/questions** to Motions? (text?)	
P10	How **useful** do you think the **direct interlink of Eos and Polar** is? (1-5)	
P11	How **necessary** do you think this interlink is? (1-5)	
P12	What are your **further comments/questions** to this? (text?)	

P13	How would you **rate Polars user interface**? (1-5)	
P14	Why would you rate it this way? (text)	
P15	How would you **rate the overall user experience**? (1-5 &/ text?)	
P16	How would you rate the **usefulness of Polar in general for political groups**? (1-5 &/ text?)	
P17	If **A3:** Could **you see the use** of Polar **in your political group**? Why (not)? (y/n & text)	
P18	What do you think is missing for **Polars** goals? (text)	
P19	Are there any **remaining comments/questions**? (text)	