Bern University
of Applied Sciences
Engineering and Information Technology

Master's Thesis

Master of Science in Engineering (MSE)

A Mobile Application for Boardroom Voting

Philémon von Bergen

Advisor: Prof. Dr. Reto E. Koenig
Bern University of Applied Sciences

Expert: Prof. Dr. Ulrich Ultes-Nitsche
Université de Fribourg

Date: February 7, 2014

Abstract

This document is part of a master's thesis, which realized a ready-to-be-used application for e-voting covering the special topic of boardroom voting. The application, running on a mobile device, allows spontaneous verifiable and secure e-voting for a small group of people present at the same location. It is built upon an existing cryptographic protocol specially designed for the mentioned purpose. No central infrastructure is required, thus only network enabled mobile devices are needed in order to run a vote. The realized application is highly influenced by the results of multiple usability studies.

Acknowledgment

I would like to thank everybody who contributed in one way or another to this master's thesis. First of all, I would like to thank the e-voting group of the RISIS institute for the support in my master studies. A special thank to Reto Koenig, who coached me in the different projects I had to realize and especially for his advices in this master's thesis.

I further want to thank Jürg Ritter for the work done together on this project and for the pertinent discussions we had about different topic related to this work.

I also would like to thank Ulrich Ultes-Nitsche who accepted to be the external expert evaluating this thesis.

I very much appreciated the help received from my mother, Lydia, who accepted to read this document and to note the spelling mistakes and other errors.

A special thank goes to my wife, Marlyse, who supported me during my whole master studies and motivated me to move forward.

Finally, a special thank to you, my reader, for being interested in my work and for being indulgent with my sometimes not so good English.

Statutory Declaration

I hereby declare having done this master's thesis myself without any unauthorized help. All information sources that strongly helped me in my work are fully referenced in this document or directly in the source code.

Title of the thesis:	A Mobile Application for Boardroom Voting
Firstname, Lastname:	Philémon von Bergen
Date, Place:	February 7, 2014, Biel, Switzerland
Signature:	

Contents

List of Figures

List of Tables

1. Introduction

"Government of the people, by the people, for the people" is the way Abraham Lincoln (1809-1865) defined the now most widespread political regime in the world: democracy. This idea exists since Antiquity already and has not changed a lot since then. From the beginning of democracy, it seemed necessary to allow people to give their opinion on some topics. The way of doing it changed a lot over time, especially in the last century, due to the technical and societal development.

During the past years, the research field on electronic voting (hereinafter abbreviated as e-voting) has become active and many e-voting protocols have been published. Their focus was mainly directed on large scale voting, thus requiring a central infrastructure collecting all the votes. Most of these protocols take advantage of this required infrastructure and base much of their security and computation requirements on it.

The present master's thesis however focus on a different type of e-voting scheme, namely boardroom voting. In boardroom voting, a small number of participants vote at the same location and at the same time. Boardroom voting could be achieved with a centralized protocol. However, the evolution of computer science towards mobility tends to promote spontaneity. So, having an ad-hoc boardroom voting system running on mobile device and usable without any preparation could really be welcome. This however generates new challenges in terms of security because there is no central trusted party anymore.

The e-voting group of the Research Institute for Security in the Information Society (RISIS) of the Bern University of Applied Sciences (BFH) is active in this field since a few years. This group has decided to go further into boardroom voting protocols. So, two master's theses have been realized in this context. Their goal was to develop an application for mobile device providing an ad-hoc system for boardroom voting. In one of these master's theses, the student, Jürg Ritter, implemented a protocol that initially was designed for a centralized voting scheme, but which could be adapted to boardroom voting. This protocol is described in *"A Secure and Optimally Efficient Multi-Authority Election Scheme"* written by R. Cramer, R. Gennaro, and B. Schoenmakers [2]. The obtained results can be found in Jürg Ritter's documentation [9].

In the second thesis, which is described in the present document, the protocol implemented was especially designed for boardroom voting. This protocol is described in the paper *"A Fair and Robust Voting System by Broadcast"* published by D. Khader, B. Smyth, P. A. Ryan, and F. Hao [7]. A collaborative work was done by the two students on the user interfaces in order to reuse them in their own project.

Outline This document consists of two main parts. The first one is dedicated to people who are only interested in the result of this thesis. The second part is the project report containing all the details of the implementation. Next chapter is a general chapter introducing some basis required to understand both parts.

The first part describes in detail the protocol that was implemented. A second section comments the results obtained of a study about usability of a boardroom voting application. Then this part ends with a discussion on usability and on efficiency of the protocol.

The second part focuses on three aspects. The first one is the organization of the thesis. The second one is the implementation of a mobile application for boardroom voting. The third aspect is the results obtained in this project. Finally, the last chapter is dedicated to improvements that can be made in the application.

2. Background and Related Work

This chapter presents the problems that apply in e-voting and lists some desirable properties for this field of application. Then follows an overview on cryptographic primitives used to satisfy security requirements for secure e-voting. The focus is put on primitives that are important for the protocol that was implemented in this project. Then, a description is given on how the protocol works.

2.1. Desirable Properties for Secure E-voting

In our world, more and more tasks can be done electronically. Even bank transactions can now be requested on a computer since e-banking became popular in the last few years. Of course, a desire appeared to do the same with voting. Although the research field is active since a few years, there are not a lot of working and secure e-voting systems. The reason for that must be searched in the properties that are commonly recognized as security requirements for e-voting. Here follows a non-exhaustive list of some of them:

- *Eligibility*: only legitimate voters can submit a vote, and only one.
- *Integrity*: once a vote cast, it cannot be modified or destroyed.
- *Accuracy*: invalid votes are not counted at tally phase.
- *Vote privacy*: nobody can say how another voter has voted.
- *Anonymity*: nobody can say whether someone else has voted or not.
- *Receipt-freeness*: the voter gets no information that could be used to prove to a coercer that they have voted in a specific way.
- *Fairness*: no results can be computed before voting period is over
- *Coercion resistance*: nobody can force someone to vote in a specified way. This implies that a coercer cannot know if the voter voted as asked.
- *Robustness*: if a small number of people collude, they cannot compromise one of the properties mentioned here.
- *Individual verifiability*: a voter can check if their ballot has been counted as they cast it.
- *Universal verifiability*: everyone can verify the correctness of a vote outcome.

These properties are inspired from the publication of Rolf Haenni and Oliver Spycher [5] and the one of Delaune and Kremer [3]. As one can see, the requirements for secure e-voting are high, even higher than for paper voting or mail voting. For example, individual verifiability does not apply to mail voting, since the voter has no way to know if their vote has been counted. Currently, none of the used e-voting systems fulfills all these requirements neither does the product developed in this project. The goal however is to fulfill as many as possible.

2.2. Cryptography Basics

In this section, some cryptographic primitives are briefly introduced. These primitives, with a lot of others, are widely used in e-voting, but also in other fields of application. This section only focuses on the building blocks that are used in the protocol that was implemented in this project.

2.2.1. Discrete Logarithm Problem

Public key cryptography relies on mathematical problems for which there is no known efficient solution. They are called one way functions. These functions are easy to compute for every input, but they are difficult to invert. This means that it is hard to find the preimage of a given image. *Easy* and *hard* must be understood in term of computational effort. *Easy* can be computed in polynomial time while *hard* cannot.

In the field of cryptography, there are two well-known and widely used such functions. The first is the prime number decomposition. It is easy to compose a number with given prime factors. However, if the number is large enough, it is considered to be hard to find the prime factors of a given random number. This problem is the base of the RSA encryption and signature schemes.
The second widely used problem is the *discrete logarithm problem*. Solving this problem means find an integer k solving the equation $b^k = g$, where b and g are elements of a mathematical group. This is believed to be difficult and time consuming for large numbers. There is no known efficient algorithm yet. Most of the asymmetric cryptography schemes that are used in this project base on the assumption that this problem is not solvable for large numbers. A typical example, that is however not used here, is the Elgamal crypto system.

2.2.2. Homomorphic Encryption

A homomorphic function is a function for which following condition holds. Let us define two mathematical groups $(X, \bigoplus)$ and $(Y, \bigodot)$ with their corresponding operation. Function $f : X \to Y$ is homomorphic for operations $(\bigoplus, \bigodot)$ if and only if:

$f(m_1) \bigoplus f(m_2) = f(m_1 \bigodot m_2)$

This property is interesting for encryption schemes, particularly in e-voting. A homomorphic encryption scheme allows to do computations on encrypted ballots in the same manner that they would be done with decrypted ballots. In e-voting, the interest of such a scheme is that ballots must not be decrypted before performing an operation, such as tallying. This is very useful the votes secret.

In some cases, the homomorphism is obtained through multiplication, but for summing up votes, addition is needed. In such cases, this operation can be obtained by putting the interesting value in a exponent, using the following property:

$a^x \cdot a^y = a^{x+y}$

In this case, the interest lies in the addition of x and y but a multiplication homomorphism is used. A random value a can be used as base for the exponentiations. With this trick, the sum is obtained in the exponent of the result.

In order to get rid of the base a, a discrete logarithm must be computed on the obtained result. As mentioned earlier, this is considered to be too inefficient on big numbers. For this case in e-voting, one can try out every possible solutions because they are not too numerous.

2.2.3. Zero Knowledge Proofs

Zero knowledge proofs allow a prover to prove to a verifier that a given statement is true, without revealing more information about it. A secret is needed to forge the proof, thus nobody else than the owner of the secret is able to create this proof. The zero knowledge proof is a probabilistic proof. It can be explained as follows. Peggy (the prover) wants to prove to Victor (the verifier) that she owns a key. They stand in front of a cave shaped as circle. In the middle of the cave, at the opposite of the entrance, there is a closed door. This door can only be opened with the key that Peggy claims to own. So, Peggy enters in the cave, randomly chooses to go left or right and goes to the door. During this time, Victor stays outside, so he does not see which path is taken by Peggy[1].

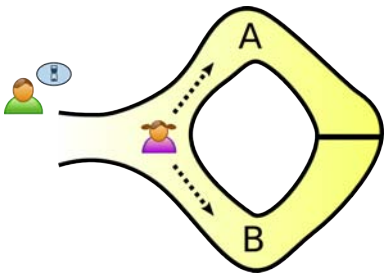

Figure 2.1.: Zero knowledge proof: Peggy chooses a way

Then Victor enters in the cave and shouts the way he wants Peggy to come back.

[1]Images and information from http://en.wikipedia.org/wiki/Zero-knowledge_proof

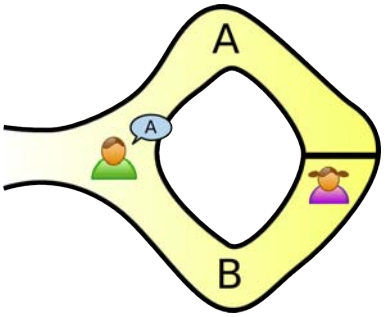

Figure 2.2.: Zero knowledge proof: Victor enters

If Peggy took the right side, and Victor asks her to return through the left side, Peggy will have to open the door with the key she owns in order to fulfill Victor's request.

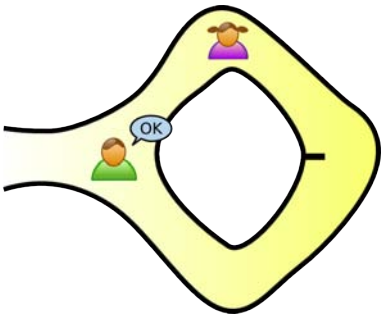

Figure 2.3.: Zero knowledge proof: Peggy returns the indicated way

If Peggy is lying about the ownership of the key, she will not be able to go through the door. However, if she went right, and Victor asks her to return right, she will be able to fulfill the request. As one can see, there is 50% probability that she does not own the key but can satisfy the demand anyway. In order to reduce this probability, this scenario can be repeated many times. So, Peggy is able to prove to Victor that she owns the key without showing it.

As one can see, this proof needs an interaction between the prover and the verifier. There is a variant of this proof that does not require this interaction. It is called *non interactive zero knowledge proof*.

Non Interactive Zero Knowledge Proofs

In many cases, a prover should be able to compute a zero knowledge proof without having to interact with a verifier. This is possible by replacing the challenge normally asked by the verifier with a cryptographic hash function. The content hashed and used in the proof is publicly known by the prover as well as by the verifier. So, this hash has the same functionality as the challenge in interactive proofs. This technique, also called Fiat-Shamir heuristic, was described by Fiat and Shamir in 1987 [4]. This scheme has the advantage that everybody can verify a proof once published and not only the verifier that sent the challenge, as that would be the case in the interactive variant.

There are various types of proofs. Here, only those used in this protocol are described. Moreover, this section only gives an overview on what these proofs do. They are described in more detail in chapter 3.

Proof of knowledge of a discrete logarithm This proof allows the prover to demonstrate that they know the preimage of a discrete logarithm. If they publish $y = g^x$, they can prove that they know x without revealing it. In other words, they know the discrete logarithm of g^x. Here, g is a generator of a cyclic group.

Proof of validity The prover has to choose one preimage out of a set of authorized preimages. Then, they compute the image corresponding to the chosen preimage. This proof then allows them to show that the computed image is the image of one preimage contained in a set of authorized preimages. By doing this, nor the preimage neither the image is revealed. This proof is not a standard proof, but is more an *OR* combination of proofs. A correct proof is simulated for each unchosen preimage, and they are then all combine together with the proof corresponding to the selected preimage. In the present case, this proof is used to show that the submitted encrypted vote is one of the authorized votes.

Proof of equality between discrete logarithms This proof allows a prover to demonstrate that two discrete logarithms are equal, in other words, that the prover used the same secret for computing two different exponentiations. For example the prover want to prove that they used x to compute $a = g^x$ and $b = h^x$ so that $\log_g a = log_h b$.

2.2.4. Commitment

A commitment scheme allows one to commit to a value and keep it secret to others during a certain time, revealing it later. A commitment cannot be modified once computed. Once the commitment and the committed value published, everybody can check if the commitment corresponds to the committed value. Commitments always take place in two phases:

1. first, the value is chosen, the commitment is computed and then published
2. second, the committed value is published, and the commitment can be verified

Commitments have two main properties, hiding and binding. A commitment is called perfectly binding if the committed value cannot be computed before the second phase, no matter how much computing power is used. For a perfectly binding commitment, it is not possible to change the value of the commitment, no matter how much computing power is used. Instead of being *perfectly* hiding or binding, these properties can be *computationally* hiding and biding. This means that these properties are guaranteed as long as only limited computational power is available.

2.2.5. Threat Model

In the security branch of computer science, a threat model describes a set of security aspects that are considered for a specific scenario. In other words, it defines the possible attacks that could be realized and specifies the assumption under which a product is considered to be secure.

2.3. Related Work

As said before, another thesis was realized concurrently by Jürg Ritter [9]. Its goal was the same as this thesis with the only difference that another protocol had to be implemented.

The University of Luxembourg already implemented the protocol used in the present thesis, but the implementation only supports computers.

The Technische Universität Darmstadt is also planning to implement a mobile app running different protocols. Their app should support various types of votes (1-out-of-n, k-out-of-n, yes-no, ...) and should automatically choose the most efficient protocol depending on the type of vote that will be done.

Part I.

Theory

3. Protocol to Implement

The protocol that was implemented was published in a paper titled *A Fair and Robust Voting System by Broadcast* written by Dalia Khader et al in 2012 [7]. In fact, this protocol is an extension of another published in 2010 in *Anonymous Voting by Two-Round Public Discussion* by Hao, Ryan et Zieliński [6]. This last mentioned paper proposes a decentralized voting protocol in two phases. However, it contains two disadvantages, namely the lack of robustness if one of the voters does not participate in the second phase and the lack of fairness since some of the voters can compute a partial result before others.

The second publication proposes solutions to these problems by adding two new phases. The first, solving the problem of fairness, is a round where each participant makes a commitment for the vote they choose and publishes it before publishing the vote itself. That avoids that a participant receives the votes and computes a partial result in order to adapt their own vote. The second phase is a recovery phase allowing to recover if a voter does not submit a vote, solving the robustness issue.

This chapter merges both mentioned publications. Its goal is to explain the whole protocol developing the mathematical formulas in order to verify the correctness and facilitate its understandability. In this chapter, no distinction is made between the elements of the first published protocol and the improvements of the second. In this part of the document, the devices running the protocol are assumed to be secure.

3.1. Properties of the Protocol

The goal of the protocol is to obtain a decentralized voting system, what means a system without any central infrastructure. Only voters participating to the vote are involved in the protocol and no other third party. This property implies some difficulties that are not present in a centralized voting system. In the original papers, some aspects are analyzed in more detail, namely:

- *Perfect ballot secrecy*: the choice made by a voter is never disclosed, this means that nobody can know what a participant has voted. In some cases, for example if there are only two voters, some information can be obtained from the final result. Such cases can however not be avoided here.
- *Self-tallying*: since no third party is involved, the tally must be done by the voters themselves.
- *Fairness*: nobody can compute partial results before having voted themselves. More precisely, this means that the computation of partial results would theoretically be possible, but the person doing that would not be able to change their vote.

- *Dispute-freeness*: this property is satisfied if each voter (and maybe even a third party) can check that the protocol was respected by all the voters.
- *Robustness*: a corrupt voter cannot make the protocol fail.

These properties are analyzed in more detail in the section 3.4.

This protocol bases on the assumption that there is a public and authenticated communication channel used to exchange the needed messages. This notion of public channel ensures that every participant can see what happens on the network, in order to avoid private message exchanges between two participants, for example the submission of two ballots or other form of collusion. The authentication property avoids that a voter sends a message on behalf of somebody else. So, the property of *dispute-freeness* can be assured.

3.2. The Protocol in Details

The protocol described in the publication is composed of three main rounds.

- *Setup round*: this is a preparation phase during which each voter chooses keys and publishes those that must be known by the other participants.
- *Commitment round*: during this phase, each voter chooses the option or candidate they want to vote and commits themselves to this choice. Only the commitment is published at this stage.
- *Voting round*: during the third phase, the vote of each participant is published.

After these three phases, the result of the vote can be computed. If one of the voters left the protocol run after having participated to the first or the second phase but before participating to the third, then a recovery round must be initiated. This is necessary because the keys of the voters are combined during the setup round. If a voter does not submit a ballot, the result cannot be computed as long as the key of this participant is taken into account. This recovery round only computes a cancelation key for the user that did not submit a ballot.

The following sections present the mathematical computation done in each round.

3.2.1. Initialization

First, there are some initializations needed:

p and q are two prime numbers such as $p = 2q + 1$.

The computations are done in multiplicative cyclic group $(G, \cdot)$ of order q which is a subgroup of the multiplicative group Z of order p (Z_p^*) and where the Decisional Diffie Hellman problem is intractable in a polynomial time. g is a generator of G. The $n \in \mathbb{Z}_q$ participants of the protocol (identified with the index i) must agree on (G, g) and p and q. $\mathbb{Z}_q$ is an additive modular group of order q.

To make the understandability easier, the explanations given here consider that a vote $v \in \{0, 1\}$ where 0 means choice *no*, and 1 choice *yes*. Explanations how the protocol can be extended to support multiple candidates or options will be given later.

In order to make it more readable, modulos have been omitted. In reality, computations are done $mod\ p$ and $mod\ q$ in the exponent.

3.2.2. Setup Round

Each voter $i \in n$ chooses a private value x_i and computes the corresponding public value a_i which is a commitment to x_i. This commitment is published.

$$x_i \in_R \mathbb{Z}_q$$
$$a_i = g^{x_i}$$

Proof of knowledge of x_i

At this stage, i must also prove that they know the value x_i, in order that the other participants can verify if i is really the owner of the private value they claim to have. This can be done with a *proof of knowledge of discrete log*.

Initialization

- H: cryptographic hash function on which the participant agreed

Proof

- i chooses $v \in_R \mathbb{Z}_q$
- they compute $z = H(g, g^v, g^{x_i}, i)$
- they compute $r = v - x_i z$
- finally, they publish (g^v, r)

Verification

- the verifier checks if g^v and $g^r a_i^z$ are equal

Demonstration

$$\begin{aligned} g^v &= g^r \cdot a_i^z \\ &= g^r \cdot g^{x_i z} \\ &= g^{v - x_i z} \cdot g^{x_i z} \\ &= \frac{g^v}{g^{x_i z}} \cdot g^{x_i z} \\ &= g^v \end{aligned}$$

The figure 3.1 resumes the setup round. Red values are private values and the green ones are published values.

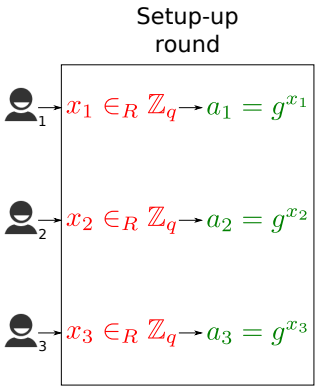

Figure 3.1.: Setup round

3.2.3. Commitment Round

Each voter $i \in n$ computes h_i as follows:

$$h_i = g^{y_i} = \frac{\prod_{j=1}^{i-1} a_j}{\prod_{j=i+1}^{n} a_j} = g^{(x_1+\ldots+x_{i-1})-(x_{i+1}+\ldots+x_n)} \tag{3.1}$$

Moreover, they choose the option they want to vote and compute

$$b_i = h_i^{x_i} \cdot g^{v_i}$$

where v_i is their vote and $v_i \in \{0, 1\}$. b_i is not published at this stage in order to avoid computation of partial results. h_i is not published either.

Proof of validity of v_i

In order to proof that their vote is valid, i has to compute *a disjunctive proof of equality between discrete logarithms*. In order to compute this proof, the terms $(g^{x_i}, g^{x_i y_i} g^{v_i})$ can be considered as a pair (a, b).

Initialization

- H: cryptographic hash function on which the participant agreed
- $(a, b) = (g^{x_i}, g^{x_i y_i} g^{v_i})$ where $g^{v_i} = 1$ ("no" vote) or $g^{v_i} = g$ ("yes" vote)

Proof

- For each $k \in \{0, 1\} \backslash v_i$
- the voter chooses $c_k \in_R \mathbb{Z}_q$, $s_k \in_R \mathbb{Z}_q$, $w \in_R \mathbb{Z}_q$

- they compute

$$\begin{aligned} a_k &= \frac{g^{s_k}}{a^{c_k}} \\ b_k &= \frac{h^{s_k}}{\left(\frac{b}{g^k}\right)^{c_k}} \end{aligned}$$

- For the vote v_i, they choose $w \in_R \mathbb{Z}_q$ and compute the witness: $a_v = g^w$ et $b_v = h^w$
- Challenge: $c_v = H(a, b, a_0, b_0, a_1, b_1) - \sum_{k \in \{0,1\} \setminus v_i} c_k$
- Response: $s_v = w + x_i \cdot c_v$
- they finally publish the proof (a_k, b_k, c_k, s_k) respectively (a_v, b_v, c_v, s_v) for all $k \in \{0, 1\}$

Verification

- Given (a, b) et $(a_0, b_0, c_0, s_0, a_1, b_1, c_1, s_1)$
- For all $k \in \{0, 1\}$, the verifier checks if $g^{s_k} = a_k \cdot a^{c_k}$ and $h^{s_k} = b_k \cdot (b/g^k)^{c_k}$
- they also check if $H(a, b, a_0, b_0, a_1, b_1) = \sum_{k \in \{0,1\}} c_k$

Demonstration
For $v = v_i$:

$$\begin{aligned} g^{s_v} &= a_v \cdot a^{c_v} \\ g^{w + x_i \cdot c_v} &= g^w \cdot a^{c_v} \\ &= g^w \cdot g^{x_i c_v} \end{aligned}$$

$$\begin{aligned} h^{s_v} &= b_v \cdot \left(\frac{b}{g^k}\right)^{c_v} \\ h^{w + x_i \cdot c_v} &= h^w \cdot \left(\frac{b}{g^k}\right)^{c_v} \\ g^{y_i w + x_i y_i \cdot c_v} &= g^{y_i w} \cdot \left(\frac{b}{g^k}\right)^{c_v} \\ g^{y_i w + x_i y_i \cdot c_v} &= g^{y_i w} \cdot \left(\frac{g^{x_i y_i} \cdot g^v}{g^k}\right)^{c_v} \text{ with } g^k = g^v \end{aligned}$$

For each $k \in \{0, 1\} \setminus v_i$:

$$\begin{aligned} g^{s_k} &= a_k \cdot a^{c_k} \\ &= \frac{g^{s_k}}{a^{c_k}} \cdot a^{c_v} \end{aligned}$$

$$
\begin{aligned}
h^{s_k} &= b_k \cdot \left(\frac{b}{g^k}\right)^{c_k} \\
&= \frac{h^{s_k}}{\left(\frac{b}{g^k}\right)^{c_k}} \cdot \left(\frac{b}{g^k}\right)^{c_k}
\end{aligned}
$$

This proof contains a commitment for v_i because b_i is hashed together with the other values of the proof. So, this proof has two roles. It allows to proof the validity of the vote and allows the participant to commit themselves to their vote. This proof is published in this round and works also as commitment.

One also can see that this proof is easily extensible to more that two options.

The figure 3.2 resumes the two first rounds. As before, red values are private values and the green ones are published values. The black ones are computed values.

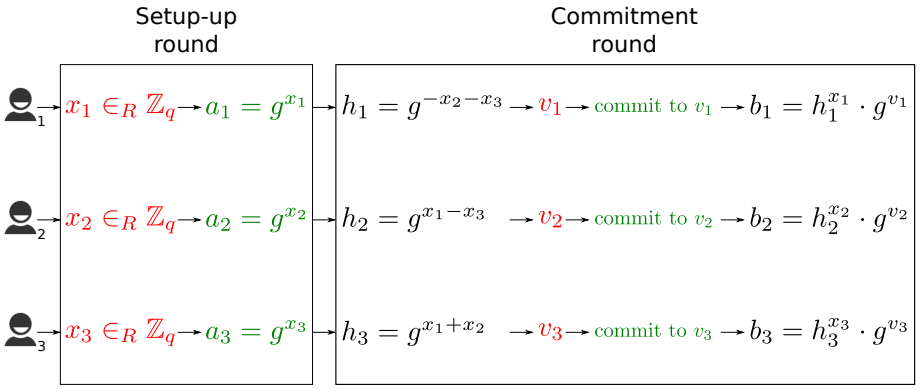

Figure 3.2.: Until commitment round

3.2.4. Voting Round

The voting phase simply consists of publishing b_i that was computed before. Since a commitment to v_i has been published in the previous round, the participant is no more able to modify their vote. They have to publish the b_i (containing v_i) that they generated for the proof of the previous round. This allows to ensure the fairness property.

The figure 9.3 resumes the different rounds until the voting round.

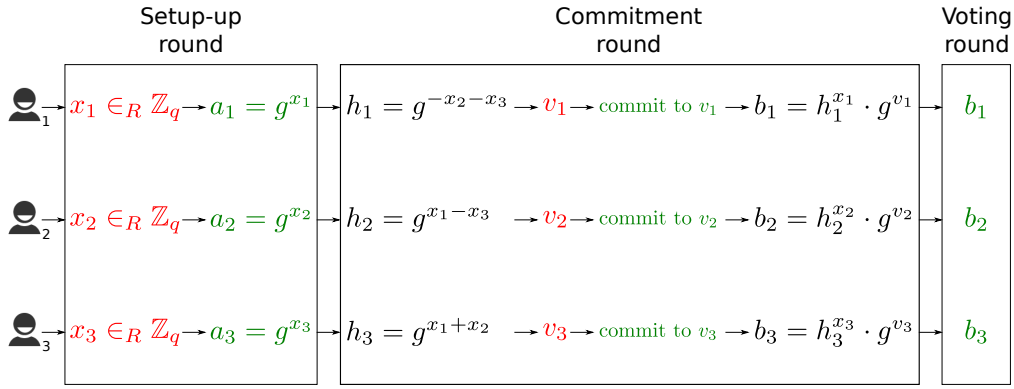

Figure 3.3.: Until voting round

3.2.5. Tallying

At this stage, the proofs have to be verified. If they are correct, the result can be computed in a homomorphic manner:

$$\prod_{i=1}^{n} b_i = \prod_{i=1}^{n} h_i^{x_i} g^{v_i} = \prod_{i=1}^{n} g^{x_i y_i} g^{v_i} = g^{\sum_{i=1}^{n} v_i} \tag{3.2}$$

where $\gamma = \sum_{i=1}^{n} v_i$ is the sum of 1 votes. The 0 votes can be computed with $n - \gamma$.

The discrete logarithm g^{γ} can be computed because of the relatively small value of γ. An algorithm like *baby-step giant-step* can be used in this case.

Demonstration

The equation 3.2 can be explained as follows:

$$\prod_{i=1}^{n} g^{x_i y_i} = 1 \tag{3.3}$$

Thus,

$$\prod_{i=1}^{n} g^{v_i} = g^{\sum_{i=1}^{n} v_i}$$

From 3.3:

$$\prod_{i=1}^{n} g^{x_i y_i} = 1 \Rightarrow \sum_{i=1}^{n} x_i y_i = 0 \tag{3.4}$$

From equation 3.1 we can deduct:

$$y_i = \sum_{j=1}^{i-1} x_j - \sum_{j=i+1}^{n} x_j \tag{3.5}$$

By inserting 3.5 in 3.4, we obtain:

$$\begin{aligned}
\sum_{i=1}^{n} x_i y_i &= \sum_{i=1}^{n}\sum_{j=1}^{i-1} x_i x_j - \sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i x_j \\
&= \sum_{j<i}\sum x_i x_j - \sum_{i<j}\sum x_i x_j \\
&= \sum_{j<i}\sum x_i x_j - \sum_{j<i}\sum x_j x_i \\
&= 0
\end{aligned}$$

The figure 3.4 resumes the different rounds.

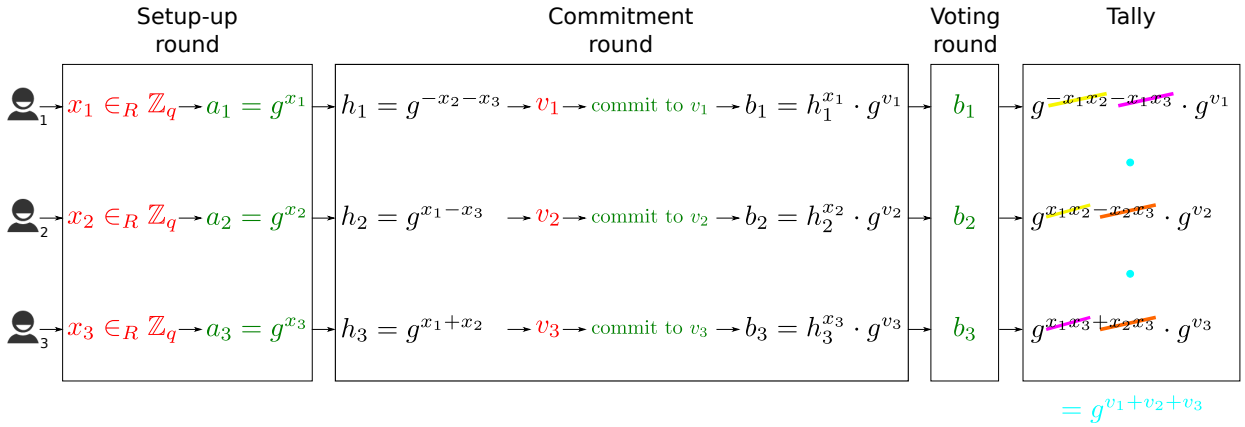

Figure 3.4.: Until tally round

3.2.6. Recovery Round

Equation 3.2 shows that if one of the participants does not submit a vote or submits an invalid vote, their corresponding b_i will be missing. Thus, the result cannot be computed correctly. To solve that, an additional round must be added where each $i \in L$ (where L is the set of participants that have submitted a valid vote and $|L| < n$) computes

$$\hat{h}_i = \frac{\prod_{j \in \{i+1,\ldots,n\} \setminus L} a_j}{\prod_{j \in \{1,\ldots,i-1\} \setminus L} a_j} = g^{\hat{y}_i} \tag{3.6}$$

they then publish $\hat{h}_i^{x_i}$ and a proof that this value was computed correctly.

Proof of correct computation of $\hat{h}_i^{x_i}$

In order to prove that i has correctly computed $\hat{h}_i^{x_i}$, they will publish a proof certifying that $\log_g g^{x_i} = \log_{\hat{h}_i} \hat{h}_i^{x_i}$. This proof is commonly called *a proof of equality between discrete logarithms*.

Initialization

- H: cryptographic hash function on which the participant agreed

Proof

- i chooses $w \in_R \mathbb{Z}_q$
- they compute $g' = g^w$ et $\hat{h}'_i = \hat{h}_i^w$
- challenge $c = H(g', \hat{h}'_i)$
- response $s = w + c \cdot x_i$
- finally, they publish $(g', \hat{h}'_i, s)$

Verification

- Given $g, \hat{h}_i, g^{x_i}, \hat{h}_i^{x_i}$ and the proof $(g', \hat{h}'_i, s)$ the verifier checks if $g^s = g' \cdot (g^{x_i})^c$ and $\hat{h}_i^s = \hat{h}'_i \cdot (\hat{h}_i^{x_i})^c$ where $c = H(g', \hat{h}'_i)$

Demonstration

$$\begin{aligned} g^s &= g' \cdot (g^{x_i})^c \\ g^{(w+c\cdot x_i)} &= g^w \cdot (g^{x_i})^c \\ &= g^{(w+c\cdot x_i)} \end{aligned}$$

$$\begin{aligned} \hat{h}_i^s &= \hat{h}'_i \cdot (\hat{h}_i^{x_i})^c \\ \hat{h}_i^{(w+c\cdot x_i)} &= \hat{h}_i^w \cdot (\hat{h}_i^{x_i})^c \\ &= \hat{h}_i^{(w+c\cdot x_i)} \end{aligned}$$

The figure 3.5 resumes the different rounds until the recovery round. Let us imagine in the present case, that voter 2 does not submit a vote.

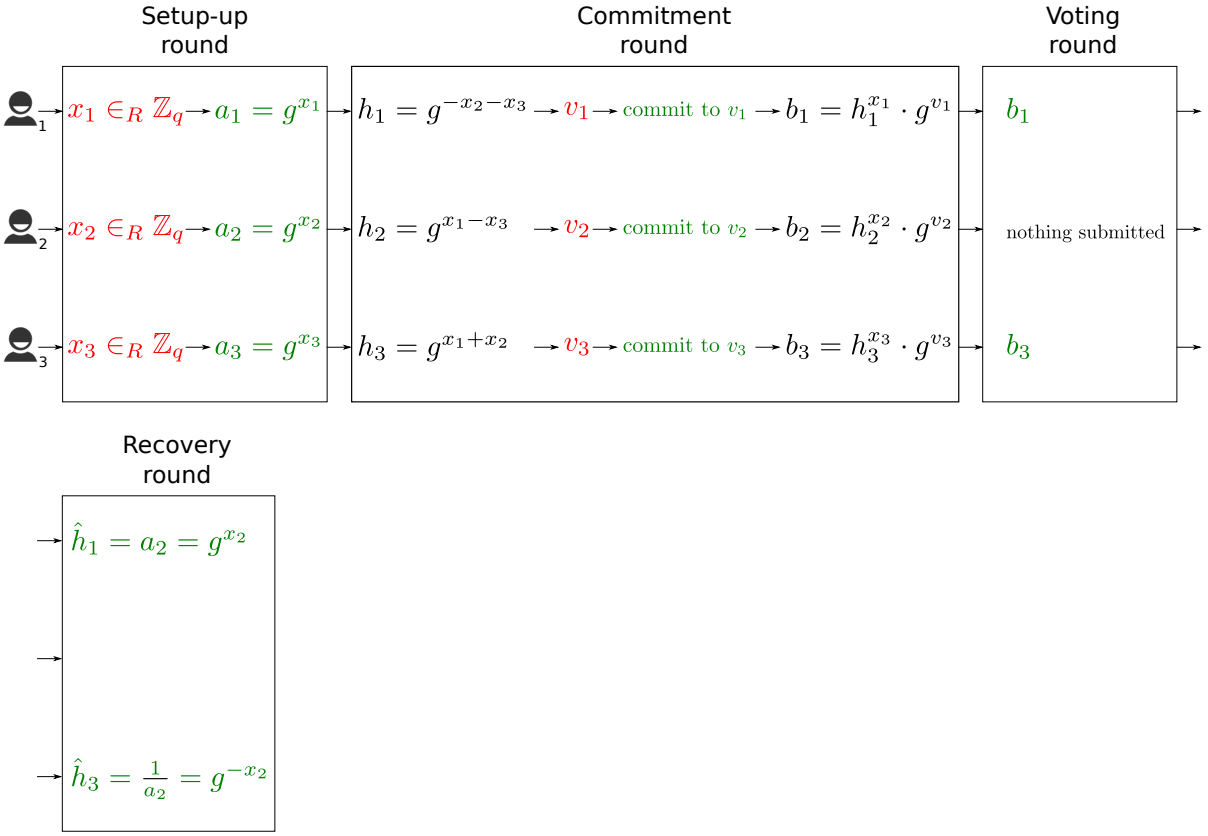

Figure 3.5.: Until recovery round

3.2.7. Tallying after Recovery Round

Once the previous explained proof verified, the L voters using the values obtained in the recovery round can compute the result as follows:

$$g^{\sum_{i \in L} v_i} = \prod_{i \in L} \hat{h}_i^{x_i} \cdot b_i = \prod_{i \in L} \hat{h}_i^{x_i} \cdot h_i^{x_i} \cdot g^{v_i} \tag{3.7}$$

Demonstration

When rewriting equation 3.7, we obtain:

$$g^{\sum_{i \in L} v_i} = \prod_{i \in L} g^{\hat{y}_i x_i} \cdot g^{y_i x_i} \cdot g^{v_i} = g^{\sum_{i \in L} x_i y_i + x_i \hat{y}_i} \cdot g^{\sum_{i \in L} v_i} = g^0 \cdot g^{\sum_{i \in L} v_i}$$

We see that following condition is satisfied:

$$\sum_{j \in L} (x_i y_i) + (x_i \hat{y}_i) = 0 \tag{3.8}$$

When considering equations 3.1 and 3.6 where we can deduct that

$$y_i = \sum_{j=1}^{i-1} x_j - \sum_{j=i+1}^{n} x_j \quad \text{(from 3.1)}$$

$$\hat{y}_i = \sum_{j \in \{i+1,\dots,n\} \setminus L} x_j - \sum_{j \in \{1,\dots,i-1\} \setminus L} x_j \quad \text{(from 3.6)}$$

we see that 3.8 is true.

So, we can conclude that $\hat{h}_i^{x_i}$ works as cancellation key for each participant that has not submitted a valid vote.

The figure 3.6 resumes the different rounds until the tally after the recovery round. Voter 2 is still out of the game.

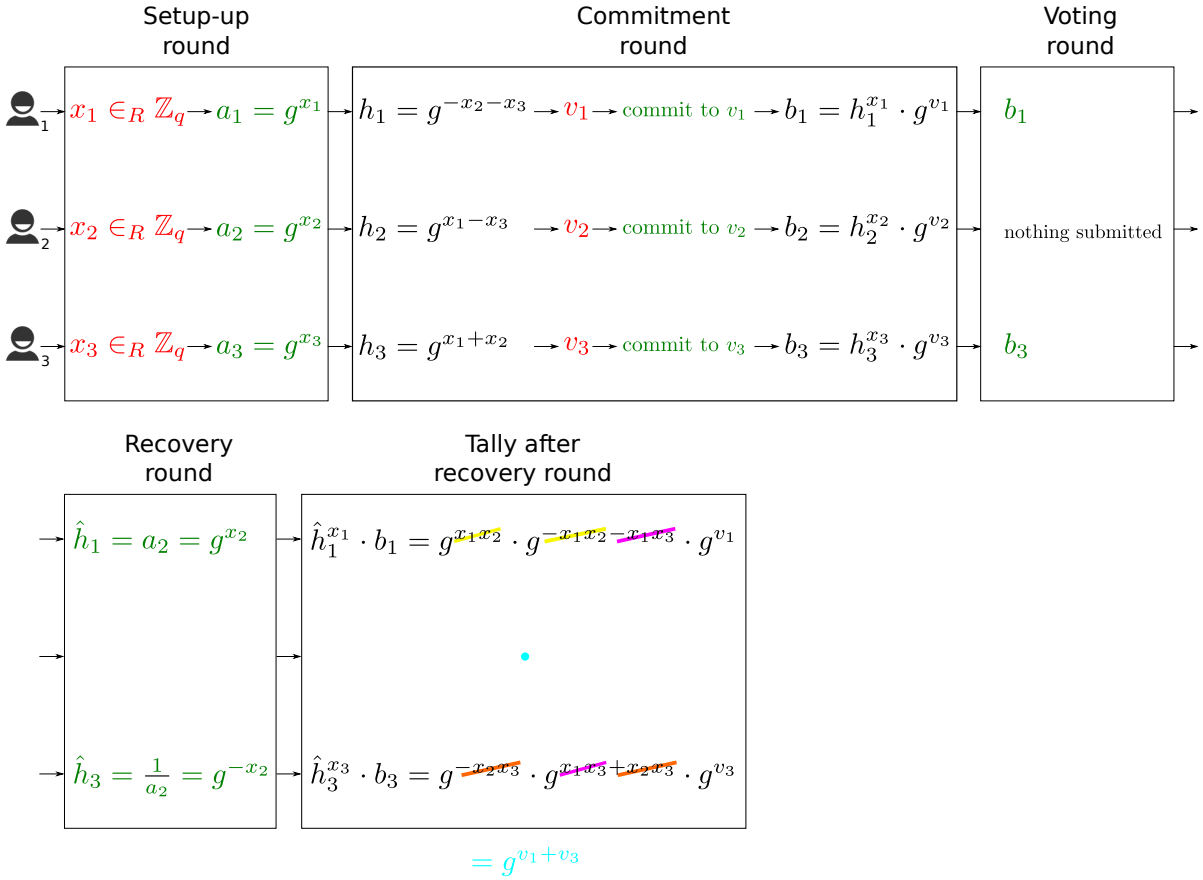

Figure 3.6.: Tally after recovery round

3.2.8. Extension to Multiple Candidates or Options

The description of the protocol above only supports a vote $v_i \in \{0, 1\}$ which corresponds to a yes/no vote or to a two candidates election. However, this concept can be extended to support a greater number of options.

The first approach would be to repeat the protocol k times in order to support an election with k candidates. At each repetition, a voter could choose to elect the candidate, or not. This variant is however not very usable, because of repetitive manipulations and waiting times.

A second approach is to adapt the protocol in order to support more candidates. Two levels are affected by this modification. First, the encryption of the vote must be adapted. Second, the validity proof computed during the *Commitment round* must be adapted. As this was already mentioned in the corresponding chapter, this proof can be easily adapted.

For the adaptation of the encryption of the vote, following method can be used. Instead of sending $g^{x_i y_i} \cdot g^{v_i}$, k different and independent generators (one for each candidate) are used. A vote would look like this: $g^{x_i y_i} \cdot \varrho_i$ where $\varrho_i \in \{g_1, g_2, ..., g_k\}$. The tally could be done as follows:

$$\prod_{i=1}^{n} g^{x_i y_i} \cdot \varrho_i = g_1^{c_1} \cdot g_2^{c_2} \cdots g_k^{c_k}$$

where c_1 to c_k are the number of votes corresponding to each of the k candidates. The problem with this method is that the g_i must be chosen in a manner that different values of the c_i do not generate the same product, in which case, the polynom would not be unique. Therefore, Baudron et al method [1] can be used. This method chooses a value m such that m is the smallest integer satisfying the condition $2^m > n$, where n is the maximum number of votes a candidate can receive. So, a vote for candidate 1 is encoded as 2^0, for candidate 2 as 2^m, for candidate 3 as 2^{2m}, and so on. So, v_i is:

$$v_i = \begin{cases} 2^0 & \text{for a vote for candidate 1} \\ 2^m & \text{for a vote for candidate 2} \\ \dots & \dots \\ 2^{(k-1)m} & \text{for a vote for candidate } k \end{cases}$$

This concretely means that a constant number of bits are reserved for each candidate to encode the number of votes this candidate received. The greatest value that can be encoded for one candidate is equals to the number of voters. Figure 9.20[1] shows an encoded vote for candidate B.

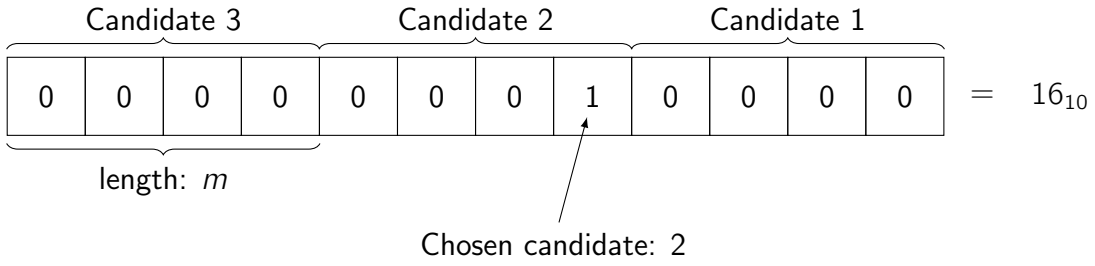

Figure 3.7.: Vote encoding using Baudron et al method

The tally can then be done as at the beginning:

$$\prod_{i=1}^{n} g^{x_i y_i} \cdot g^{v_i} = g^{\sum_{i=1}^{n} v_i}$$

[1] Image created by Jürg Ritter in [9]

where

$$\sum_{i=1}^{n} v_i = 2^0 \cdot c_1 + 2^m \cdot c_2 + ... + 2^{(k-1)m} \cdot c_k$$

Values c_1 to c_k are the number of votes for the k candidates respectively. This method allows to compute a unique polynom. This allows to do an election k-out-of-n candidates. However, the implementation in this project is limited to 1-out-of-n, because the validity proof becomes rapidly very complex for k-out-of-n elections.

3.3. Generalization of the Concept Used in the Protocol

The value b_i representing the encrypted vote of the voter i can be compared to a Pedersen commitment. Indeed, a Pedersen commitment has the form $g^x \cdot h^r$ where g and h are generators of the chosen group, r is a random value and secret of the commitment. In the case $b_i = h_i^{x_i} \cdot g^{v_i}$, h_i becomes the second generator, x_i the random value and v_i the secret of the commitment.

As seen earlier, the multiplication of all b_i causes the cancelation of the $h_i^{x_i}$. So, only $g^{\sum x_i}$ remains. This can also be applied to the Pedersen commitment. If h is chosen as h_i is in this protocol and x_i is used as random value, the multiplication of all commitments created with this scheme has the particularity to get rid of the randomization, and thus reveal the sum of the secrets contained in the commitments. It would be *self-revealing*.

This generalization could maybe be used in other fields than e-voting.

3.4. Analysis of the Protocol Properties

In the introduction of the previous section, the properties expected for this protocol were briefly described. In this section, they are analyzed in more detail based on the explanations given in the previous section.

3.4.1. Perfect Ballot Secrecy

This property means that the vote realized by a participant remains secret even if some of the other voters collude. In the protocol, each voter publishes a public ephemeral key g^{x_i}. The votes are encrypted with a key derived from the ephemeral key of each participant and with the private part of this ephemeral key (for instance, the cipher text looks like $g^{x_i y_i} \cdot g^{v_i}$). Assuming that at least one other participant (other than i) is honest, the value y_i is a random secret value, because it is computed with the x_j ($j \neq i$). As each x_j is uniformly distributed in $\mathbb{Z}_q$ and at least one x_j is unknown by the colluders, they cannot compute y_i. Thus, under Decisional Diffie-Hellman assumption, an encrypted vote cannot be differentiated of a random value.

About *zero knowledge proofs*, the proofs computed in the protocol do not reveal information on the vote. The proof computed in the setup round only reveal if a participant knows their x_i value.

The one of the commitment round also only reveals one bit of information, namely if the vote is valid or not. Finally, the one computed in the recovery round only says that $\hat{h}^{x_i}$ was computed correctly. With this information, the secrecy of the ballot cannot be broken.

The tally is obtained by the multiplication of all votes. The information known by the participants at the end of the voting period is only the final result. So, the confidentiality of the votes is kept.

3.4.2. Self-tallying

The tally of the ballot is done by each voter without any help. In the first round, each voter chooses a private key and in the second round, the corresponding public keys are combined in a way that the random values get immediately cancelled after the voting round, thus revealing the tally. The *zero knowledge proofs* ensure that the participants followed the protocol.

3.4.3. Fairness

Thank to the *commitment round*, a voter first has to choose their vote and commit themselves to this choice. By publishing this commitment before sending the vote, the voter is no longer able to change their choice afterward. The vote itself is only published once the commitment of each participant is received. So, it is not possible to compute a partial result before voting, thus, the protocol is fair.

3.4.4. Dispute-freeness

This property is ensured through the public authenticated communication channel. An attempt to send multiple votes from the same participant would be detected. Moreover, a *zero knowledge proof* allows to verify that a vote is valid and does not contain an illegal content. On the other side, the *self-tallying* property ensures the correctness of the result.

3.4.5. Robustness

Thank to the *recovery round*, a denial of service attack, in which a participant refuses to publish their vote, can be avoided. Even if this happens, the mentioned round can be used to cancel the participation of this voter, and allows to compute the result anyway.

4. Graphical User Interfaces

The graphical user interfaces (GUI) are very important for almost all types of applications. It is the part the user is confronted with. So, it is crucial that these interfaces are comprehensible and easy to use. Since multiple examples showed that security and usability do not mix well, a part of this thesis was dedicated to this subject in order to avoid this widespread deficiency and to obtain usable interfaces.

As it was already said, another student was doing a similar thesis, however with another cryptographic protocol. So, it was decided that the user interfaces study and implementation would be done together. This chapter describes the observations made.

4.1. Findings of the Study

Roles The first finding of the study was that there are two roles for a boardroom voting application. The first one is the administrator role. This member of the group has to define the vote and its properties, like the question, the choosable options and the people belonging to the electorate. This person also has to manage the flow of the process, since someone has to decide when to start and when to end the voting period. The second role is the voter role. All other members of the group just want to participate to the vote. They do not have anything special to do other than vote. The separation of these roles can also be used for the integration of a business case. Indeed, a possibility would be to ask the user of the application to pay for the administrator functionalities, while offering the voter functionalities for free.

Sometimes, the administrator could also want to vote, sometimes not. This means that under certain circumstances there are components used by both the admin role and the voter role. In order to save work, these common components should only be implemented once and used by both roles.

Review screen In the boardroom scenario, there is no trusted party, not even the administrator of the vote. This implies that the voters must be able to control if the admin sets up the vote correctly. For this reason, it is necessary to provide a review screen where each voter can accept, or not, the content of the vote (question, options and electorate). This allows a voter which disagrees with some point to manifest themselves. When someone does not accept the review, the participants can talk together to solve the problem, since, in the case of boardroom voting, they all meet together.

Since there is no trusted party, the content received is not trustworthy. The admin could send different contents to various voters. In order to avoid that, a broadcast communication channel is required in order that a message is received identically by all the voters. Boardroom voting offers

an additional protection against this type of attacks. Since all participants meet together, they can ask each other to double check the content appearing on their screen. This point is particularly important when receiving the vote properties, in other words in the review screen. This attack could also be made on the protocol messages, but it is of less interest, since the values sent are confirmed by the proofs required by the protocol.

Flow During the study, it was found that different steps are necessary in the flow.

- Vote setup: the vote setup is the step where the administrator can create a new vote and define its properties. They should be able to do it during the meeting or before the meeting and save it, in order to reuse it later.
- Network setup: the network setup step must be done during the meeting since it is dependent of the location. In this phase, all voters join a network where they can communicate together.
- Voting process: the voting process includes the review process of the vote that was mentioned earlier, the voting period, the tally and the displaying of the result.
- Archived votes: the user should have the possibility to display the result of a vote done earlier.

All these steps seemed to be necessary for a usable and secure boardroom voting application.

4.2. Obtained Interfaces

Regarding the points mentioned in the previous section, following storyboards were developed.

Vote setup The first screen shown is the view A visible in figure 4.1 (numbered arrows are jumps to other figures). On this screen, the user has to choose their role. Either they are a simple voter and want to join the electorate of an already set up vote (first button) or they want to be administrator and want to create a new vote (second button). The third button allows to access the archived votes as this will be discussed later. If the user chooses to be an administrator, view B is displayed. There, they can either choose an already set up vote or create a new one. When choosing one of these options, view C is shown where the vote's properties can be defined. There, the admin can choose to save the vote in order to reuse it later, or to start the voting process for the selected vote.

Network setup When a user chooses to join the electorate of an already set up vote or when an administrator chooses to start a voting process, view D is displayed as shown in figure 4.2. When accessing this view as administrator, the two buttons in the middle are not visible, since the administrator has to create a network group and not to join one. So, the admin can choose to use the wireless network to which their device is currently connected, or choose another network in the advanced network settings (view E). There, they can also decide to use the hotspot included in their device. By choosing one of these options, a group is created on the network to allow the devices of the voters to communicate together. For security reason, this group should be joinable for only allowed people in order to reduce risk of attacks. So, a secret must be defined and required

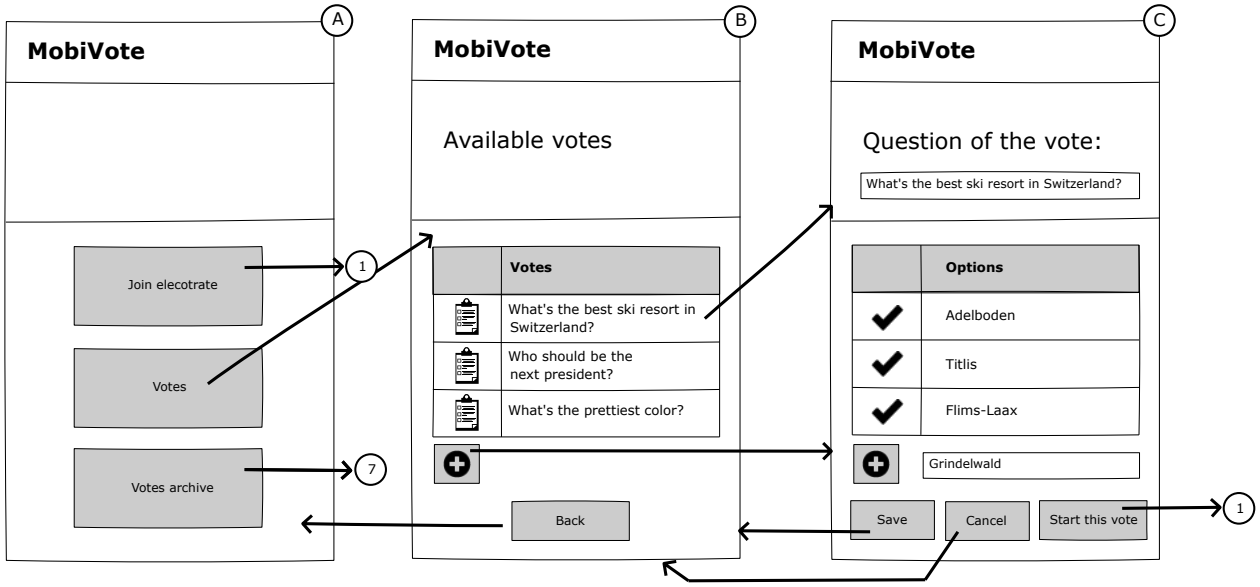

Figure 4.1.: Setup of the vote

when someone wants to join this group. This secret is fixed by the admin and displayed on the admin's device in view F.

In this view, the user also has to indicate their identification, i.e. the name they want to be displayed on the device of the other participants.

For the voter, the process on these screens is a little bit different, since they do not have to create the group but to join it. One solution for that is to choose a network and indicate the secret defined by the administrator. Another possibility is to scan a QR-code or read an NFC tag containing the secret and provided by the admin. Other possibilities could also be imagined.

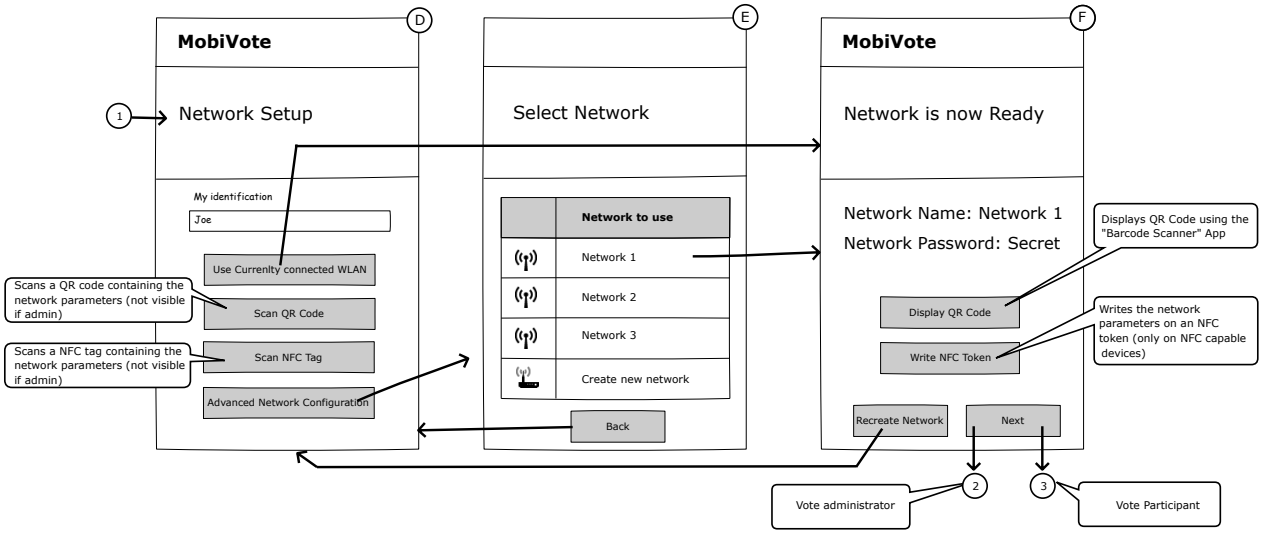

Figure 4.2.: Setup of the network

Voting process Once the network set up, view G from figure 4.3 is displayed on the admin's screen. There, the participants connected to the group are listed. The admin can choose which of

them they want to include in the electorate. Once this done, view H is displayed which is nothing else than the review screen that was mentioned in the previous section. On that screen, the admin must wait until all members of the electorate have accepted the review. Then, the administrator can start the voting period. If the admin is contained in the electorate, the voting interfaces are displayed (described later), otherwise the admin lands on a screen (view I) displaying the state of the voting process. On this view, they can manage the voting period, for example decide to cancel it or to end it. Normally, the voting period automatically ends when all participants have voted, but it can happen that, for some reasons, a voter does not submit their ballot. In such a case, the admin can decide to end the voting period.

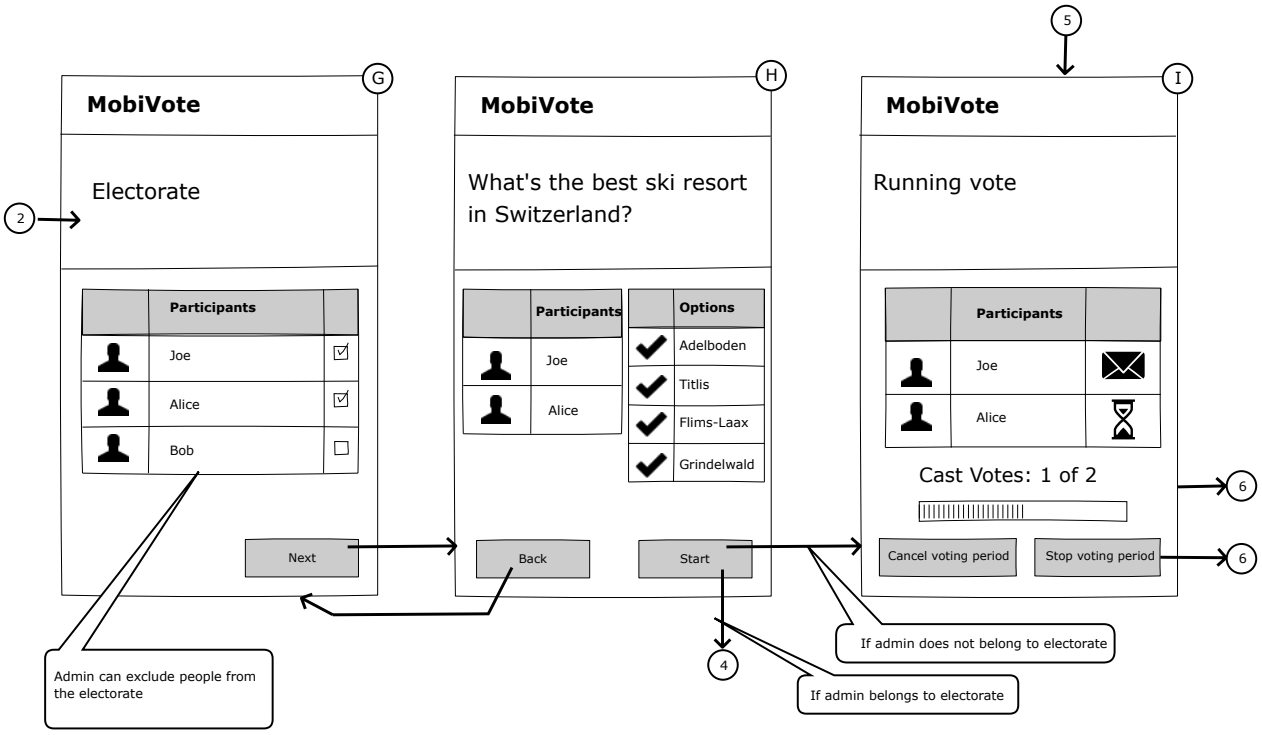

Figure 4.3.: First part of voting process for the administrator

For the voter, the first part of the voting process is slightly different than for the admin. Once connected to the group, view J (figure 4.4) is shown to the voter where they can see all participants connected to the network and which are selected as member of the electorate. When the admin has finished choosing them, view K, the review screen is shown. There, the voter has to indicate if they agree with the content of the vote or not. When everybody has accepted it and the admin decides to start the voting period, the voting interfaces are displayed.

The voting interfaces consist of two views. The first one (view L in figure 4.5) is the voting screen where each member of the electorate can choose the desired option. If the admin voted, they are redirected to view I described before, all other voters then sees view M displaying the progress of the voting process. On this screen, the voter has to wait until all voters have cast their vote or the admin decides to end the voting period.

Result and archived votes When the voting period ends, the tally process can begin on each device. Once finished, view O (figure 4.6) is displayed showing the result of the vote. On this

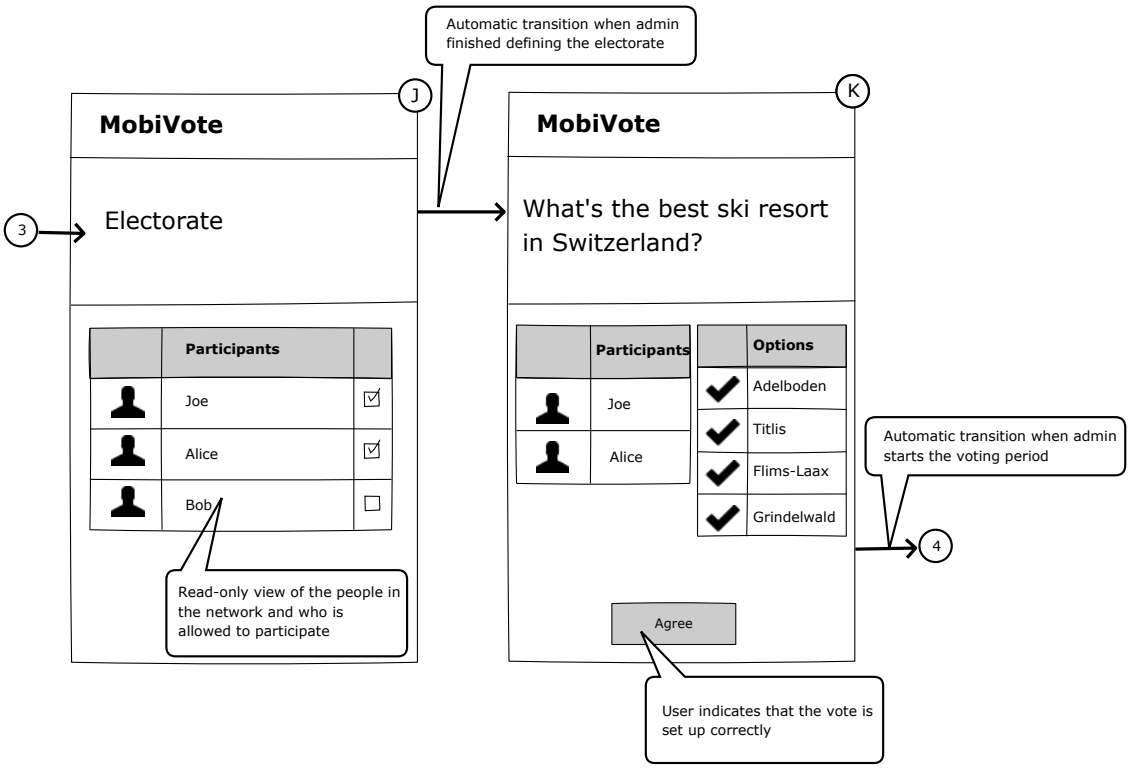

Figure 4.4.: First part of voting process for the voter

screen, a vote can be copied in order to run it again. When leaving this screen, the list of all previously run votes is displayed (view N). This view can also be directly accessed from the main screen.

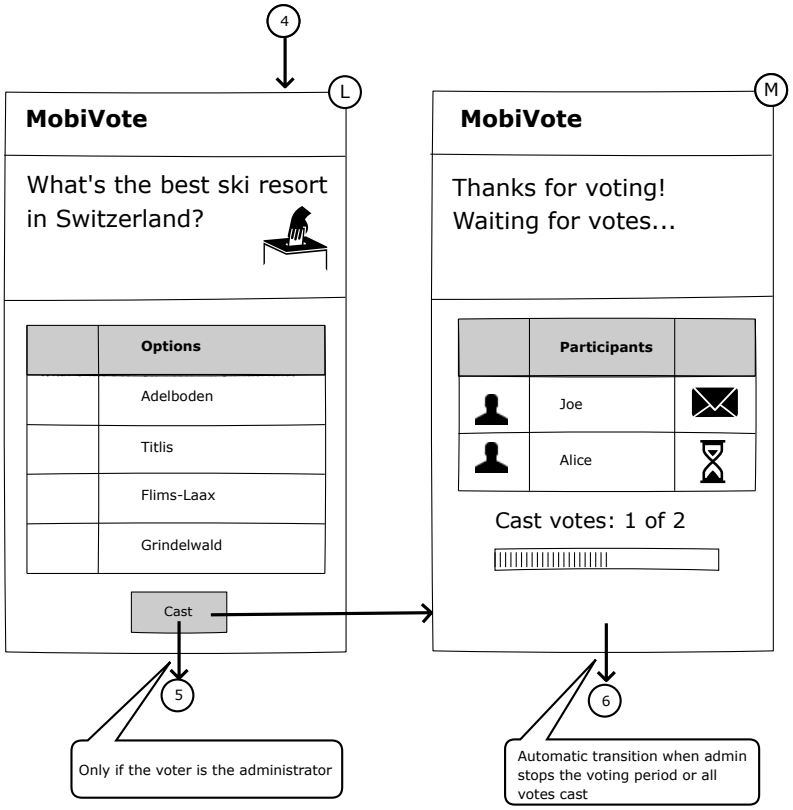

Figure 4.5.: Second part of voting process

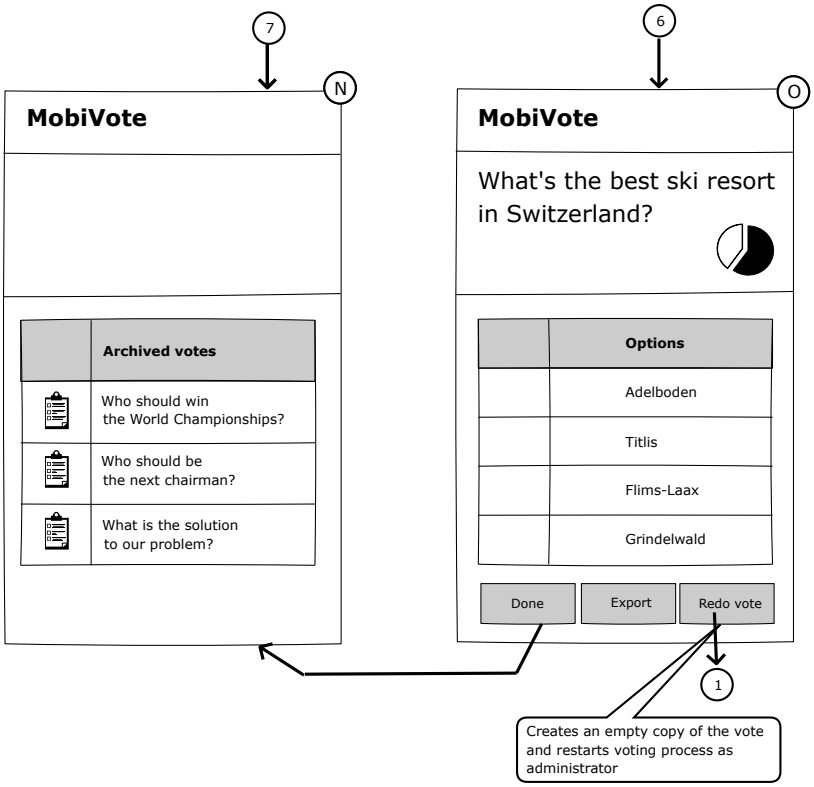

Figure 4.6.: Votes archives

5. Benchmarks

The implementation of the protocol described in section 3 and of the user interfaces described in section 4 allowed to evaluate them. This chapter describes the results in term of usability for the GUI and in term of efficiency for the protocol.

5.1. Graphical User Interfaces Benchmarks

The user interfaces were tested by people with various backgrounds. The first tries were not very good and modifications had to be made, resulting in what was presented in the previous chapter. What these usability tests showed is that making a too big separation between the admin role and the voter role is not a good idea. The first version had two different applications, one for the admin, the other for the voter. During the tests, most of the people did not know which application to use. Having two was confusing.

Another idea in the initial version was to have separated flows for the network configuration and for the voting process. First, the network had to be set up, and then the vote. This also was a bad idea since the testers always wanted to vote before configuring the network. That showed that the users were not aware of the necessity to set up the network. Thus, this part had to be moved somewhere else in the flow in order to force the user to care about it.

Once these flaws corrected and some other details adapted, the testers understood quite well how the application works.

The user interfaces were designed independently of a crypto protocol. The goal was to obtain a generic GUI usable for several protocols. At the time when the usability tests were done, the cryptographic part was not already implemented. The usability testers did not complain that the user interfaces were too complicated. The implementation of the protocol did not require major changes in the GUI. This means that the addition of the security layer could be done in a completely transparent manner, so without making the user interfaces more complicated. Two different protocols could be implemented using this GUI. So, this work proves that security and usability can be obtained together if enough time is spent on studying how to do it. The key point in this project that allowed to obtain this result was probably that the user interfaces were designed first, without caring about a specific crypto protocol.

5.2. Protocol Benchmarks

The most computationally heavy part of the protocol described is the tally process. As it stands in the protocol description in 3.2.5, the tally uses a homomorphic process to recover the result of

the vote. In the mentioned section, one also observes that there is a discrete logarithm that must be computed. For that, two solutions could be used: first, use a discrete logarithm computation algorithm like baby-step giant-step, second, list all the possible results for the vote and check if one equals the result obtained by the protocol. The first solution is not appropriate when using the Baudron et al method for the representation of each option (see 3.2.8). With this encoding, a lot of values can be encoded, but most of them do not represent a valid vote result. So, the baby-step giant-step algorithm would check a lot of value that do not represent a valid vote result. So, this algorithm is not efficient for this case.

The other solution lists all the possible results for a vote. If there are o options and p participants, the sum of the votes each option received must be equal to p (assuming blank ballots are not allowed or proposed as an option). So, permutations of length o where each position can take values 0 to p and where the sum of all positions is equal to p are allowed. The number of such permutations can be computed with following formula:

$$\binom{o+p-1}{o-1}$$

A graphical representation of this function gives:

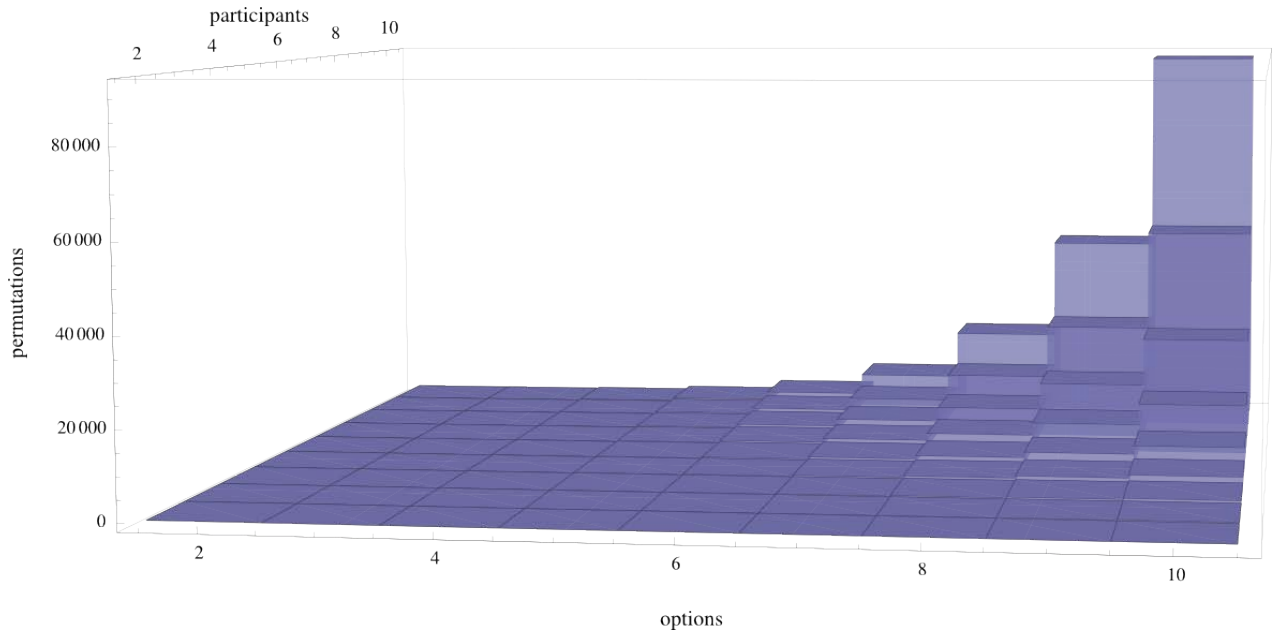

Figure 5.1.: Graphical representation of the complexity of the tally process

As one can see, the complexity grows heavily when the number of options or the number of participants increases. The standard method to compute the result would be to take each permutation representing a valid vote, to encode it as a vote v, and to compute a modular exponentiation g^v (see 3.2.5 and 3.2.8 for the details of the tally process). So, the number of modular exponents grows heavily when the number of options or the number of participants increases.

A more efficient way to compute this result is to make some precomputations allowing to transform the modular exponentiations in multiplications. The idea is to compute g^v for all v representing

a basic permutations containing 1, 2, 3, ..., up to p votes for each option independently. In this method, $o \cdot p$ multiplications are required for the precomputations and then a simple multiplication replaces the mod exp needed in the previous method. This is a good improvement, but the app still does not allow arbitrary big numbers of options or participants.

The concrete performance of the tally process depends on the device. However, one sees that this protocol, when implemented with the tally solution presented here, is usable for most of the common use-cases in boardroom voting.

6. Summary

The protocol described in the publication titled *A Fair and Robust Voting System by Broadcast* is achievable in practice. However, the tally phase is the bottleneck of this concept. An improvement on that part would be a benefit.

Concerning the security aspects, the protocol considers most of them. However, with the threat model used here, a real implementation should run on devices equipped with a secure platform in order to stay secure.

Usable user interfaces are realizable even when security is involved in the project. It however requires some reflection and investment of time to obtain something satisfactory.

Part II.

Practice

7. Organization

The present master's thesis is a graduation project in Computer Science for the Master of Science in Engineering studies at the BFH (Bern University of Applied Sciences). For this project a time frame of 720 hours was planned, which corresponds to a master's thesis of 27 ECTS. The start of the thesis was the 16th of September 2013 and the deadline is the 7th of February 2014. This chapter presents the planning that was foreseen for the project.

This project was formed of three main parts. One of these was the creation of the user interfaces. The other part was the implementation of the protocol. Added to these two parts, there was a third one consisting of the addition of the network layer. As mentioned in the introduction, another student was working on a similar project. He was also asked to develop an application for boardroom voting, however using another cryptographic protocol than the one presented in this document. Since we knew that the final product would be quite similar in the look and feel, it has been decided that the user interfaces part would be done as a joint work. The user interfaces flow was already designed in a previous project by the same students and has been discussed with their supervisors. The next section describes the planning that was made at the beginning of the project. In the following section the real progress is described and commented.

7.1. Planning

The user interfaces (UI) task was planned to be done at first. This provided the advantage that, when implementing the protocol, there was already UI available to test it. An other advantage was that the UI were not created in a hurry at the end of the project when there was no more time left, as it is often the case in computer science projects. Doing that at the beginning of the project increased the quality of the usability of the application. This task included UI creation, making usability test with people not involved in the project and testing of the UI.

For this first phase, it was planned to simulate the network layer. In a second phase, the real network layer had to be added to the application. This task included the integration of an existing network layer, the addition of some security concepts needed by the protocol and the testing.

The third phase consisted of the implementation of the crypto protocol. The first subtask was the adaptation of the UI architecture to integrate the protocol architecture. It was followed by the implementation of the cryptography required within that protocol. This phase ended after testing.

In addition to these three main phases, there was an overall testing task whose goal was to test the integration of the results of the different phases. There also was the reporting task to produce the expected documentation.

There were three milestones corresponding to the three main parts of the thesis. For the user interfaces, it was planned to have them at the end of October. One week later the network layer was planned to be implemented, since it was developed in parallel with the user interfaces. The third milestone was the end of the implementation of the protocol at mid-January, letting some time for the load tests and the documentation.

Figure 7.1 shows how much time was planned for these tasks.

Month	Sep.		October					November				December			January				Feb
Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Phase 1 : User interfaces																			
UI flow																			
Usability tests																			
Testing of UI																			
Phase 2 : Network layer																			
Add network layer																			
Security of network layer																			
Testing of network layer																			
Phase 3 : Protocol implementation																			
Preparation of protocol flow																			
Implementation of cryptography																			
Testing of protocol																			
Overall testing																			
Documentation																			

Figure 7.1.: Expected planning

7.2. Realization

Figure 7.2 shows how the working time was spread over the task. One can see that the creation of the UI took a little more time than initially planned. This can be explained by the major changes that were done after the first usability test.

The network layer implementation also took more time than expected because of the change of network layer, as this will be described in more detail in a next chapter.

For what concerns the implementation of the protocol, it took less time than estimated. This can be explained by the fact that a crypto library could be used, providing some already implemented building blocks. This resulted in the fact that an additional task, namely the implementation of a basic verifier allowing to check if the protocol ran as expected, could be realized. Moreover, in the last week, a trip to Darmstadt could be organized where discussions took place about the realization of the boardroom voting application mentioned in the related work part of section 2.

The overall testing phase was beneficial especially for the load test that could be done that revealed some bugs invisible with only few devices.

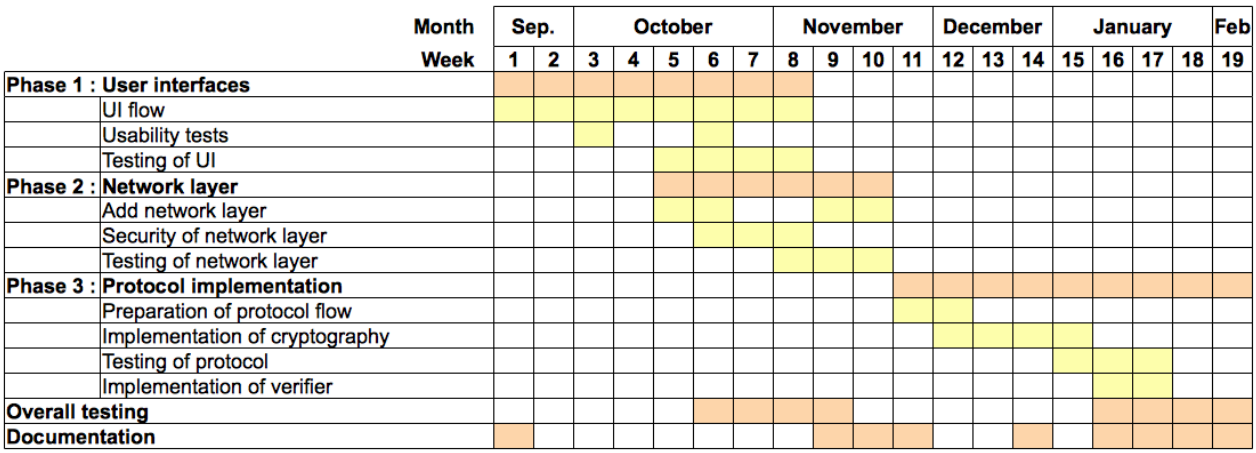

Figure 7.2.: Effective work

7.3. Source code

The source code of the application is available on Github[1]. The main project has dependencies on different library projects. Table 7.1 lists the links of the repositories and the branches to check out.

Project	Repository link	Branch
Voter App	`https://github.com/jritter/VoterApp`	hkrs12
AllJoynLib	`https://github.com/jritter/AllJoynLib`	master
ZXing	`https://github.com/jritter/ZXing`	master
MobiVote Verifier	`https://github.com/vonbp3/MobiVoteVerifier`	master

Table 7.1.: Source code repositories

[1] https://github.com/

8. Specifications

For the real implementation of the application, some choices had to be made. Some other requirements had to be fixed. This chapter lists these specifications and gives an overview of the tools used during the implementation.

8.1. Requirements for the Application

As mentioned before, the main goal for this project was to build an application for mobile devices implementing the protocol described in the paper *"A Fair and Robust Voting System by Broadcast"*. So, the resulting app had to allow a boardroom[1] to carry out a vote in a spontaneous manner without having to set up any infrastructure.

A choice had to be made for the development platform. Android was chosen. The reason for this choice is that the programming language on Android is Java and the student already had knowledge of this language and of Android programming. Furthermore, programming for Android does not require a developer license. The application had to be compatible with smartphones and tablets.

This application had to satisfy the common requirements for secure e-voting. The protocol implemented takes care of most of them. In particular, this protocol is verifiable, which means that each voter can verify that all other voters followed the protocol. Currently, if one looks in the Play Store of Android one will find some apps allowing to vote, but these apps do not necessarily respect e-voting requirement and especially not verifiability. The application developed in this project fills this gap.

By choosing smartphones and tablets as target devices, the threat model described in the first part, namely the assumption that the devices running the application are secure, is no more applicable to this setup. Android smartphones and tablets are not secure in term of display and keyboard. So, the threat model had to be adapted by considering the risk coming from the device as out of scope.

It was expected that the application allows a group of people to be able to vote (yes or no votes), or to carry out an election, with up to 25 participants, which is a quite big size for a normal boardroom. The voter had to be able to choose 1-out-of-n options proposed as response.

In order to allow the devices to communicate to each other, wireless local area network (WLAN) had to be used. This does not necessarily mean an access to the Internet but this implies to be in proximity of a wireless hotspot. If none is available, the built-in hotspot of one device can be used.

[1] By boardroom not only boardrooms of directors in a company are meant, but every type of group that should want to carry out a vote.

This last mentioned requirement implies that all people participating in the vote must be present at the same location, since a WLAN network is location bounded. This requirement can be considered as an advantage for the implementation of the specific protocol. Indeed, proximity can be used as secondary channel. So, this feature can be used to transmit some secret data (orally or visually) and can also be used as control way in term of uniformity of the content received and exchanged between the participants. For example, a participant of the vote can compare with their neighbour if the content displayed on their devices is identical.

Another goal of this project was to develop an app usable for everybody, not only for specialists of security. So, the app had to be designed in an intuitive way for the users, the texts had to be easily comprehensible. Often, usability and security do not cohabit very well. This project had to deal with that and to try to obtain the best result possible, keeping in mind that it was not developed by usability specialists.

8.2. Tools Used

In this project, different frameworks and tools were used. This section gives a little background for the most important of them.

8.2.1. Android

Android[2] is an open source operating system based on a Linux kernel designed for mobile devices as smartphones and tablets. Some other devices like television sets, games consoles, digital cameras or even cars also use this system. Android is currently developed by Google. It has a powerful developer toolkit, which allows programmer to develop apps for devices equipped with this system. The programming language used in the toolkit is Java.

Android is the most used operating system on mobile devices like smartphone and tablets (81.3%), before iOS from Apple (13.4%) and Windows Phone from Microsoft (4.1%)[3].

The system bases on five main building blocks.

1. **Activity:** an activity is a user interface. A single activity corresponds to a single screen. Each activity has a lifecycle. An application mainly contains multiple activities. The definition of the look and feel of an activity is defined through an XML file.
2. **Intent:** intents are system message notifying about events, for example hardware changes or incoming data. Intent are also used to start activities or to send data through the application.
3. **Service:** a service runs in the background and does not present a user interface. A service keeps running, independently of an activity. A typical example for a service is a music player, which runs in background.

[2] http://www.android.com/

[3] Numbers from the third trimester of 2013. Source: http://thenextweb.com/mobile/2013/10/31/strategy-analytics-android-smartphone-shipments-81-3-q3-2013-ios-13-4-windows-phone-4-1/

4. **Broadcast receiver:** an application can register itself to be notified in case of certain types of events occur. This is done by broadcast receivers. The event can be local to the application or public in the system.
5. **Content provider:** a content provider is an abstraction for data stored on the device and accessible from multiple applications. An example is the content provider allowing to access to the contacts stored on the phone.

8.2.2. InstaCircle

The crypto protocol implemented in this thesis requires a communication channel that is public and authenticated in order to allow the participants to exchange messages. InstaCircle, an Android chat application, was developed by Jürg Ritter [8] in a previous project to allow Android devices to communicate together in a public manner, what means that every participant sees all messages sent in the group. InstaCircle implements such a public channel, and on it, bases a chat application. It uses wireless LAN as network layer. So, this means that the devices that want to communicate together must be connected on the same WLAN.

InstaCirlce works as follows: the first person that connects defines a password. A network group is created. The other people that want to join the group must indicate the same password and become participants of the group. The password must previously have been transmitted by the creator to the other people on a secondary channel (orally for example). Once the group joined, a participant can see all the messages exchanged, since the messages are transmitted over UDP broadcast. The use of a password allows to have a control on who will join the group by giving the password only to people that are authorized to participate.

In order to allow to have multiple groups on the same network, messages of one group are encrypted with an AES key derived from the password of the group. If a message cannot be decrypted, it means that this message was from another group.

InstaCircle also proposes more convenient ways to transmit the password. For example, it shows up a QR-Code containing the needed connection data. It also provides an NFC functionality to transmit these data. Another feature is to use the built-in hotspot of a device to create the WLAN that is used for the message exchange. This allows to avoid to be dependent on a wireless infrastructure. However, this solution limits the number of participants since the built-in hotspot of mobile devices usually does not support a large amount of connections.

Finally, another functionality offered by InstaCircle is to recover messages if a user went offline and comes back to the group. In such a case, the user can ask the members of the group to send them the messages they missed. This recovery procedure is done over unicast.

As one sees it, this application provides a public communication channel, but it is not authenticated. A user could try to spoof their IP address and indicate the one of another participant and then send a message in their name. So, the authentication should be added in this project.

More information about this application can be found in the report of the project in which the application was developed.

8.2.3. AllJoyn

AllJoyn is a widely used alternative to InstaCircle. The reason why both are presented here will be given in a next chapter. AllJoyn[4] is an open source software developed by Qualcomm. It provides an universal framework helping developers to set up an ad-hoc, proximity-based device-to-device communication. Developers do not need to write their own code managing the connexion of various devices together anymore. Alljoyn cares about that. Core building blocks of AllJoyn are: discovery, connectivity, security and management of ad-hoc proximal networks among nearby devices.

AllJoyn is compatible with Android, iOS, Windows, Mac OS X and Linux. It supports connectivity over WLAN, LAN, and also bluetooth on mobile device. AllJoyn runs in form of a daemon on a device. This allows inter-process communication. AllJoyn implements a virtual bus connecting multiple AllJoyn daemons. The daemon can be accessed in an application using the SDK.

AllJoyn works as follows: someone, let us call them the initiator, has to create a group and to give a name to this group. The initiator's device will host the group. The group name is then advertised on the network through multicast. Another device using AllJoyn can discover the groups available on the network. If there is one, the device can connect to this group. The communication between the devices connected to the group takes place over unicast transiting over the initiator hosting the group. A message sent to the entire group first goes to the initiator, which transmits it to each participant of the group.

If multicast is disabled on the network, the advertisement of the group name cannot be done. In this case, developers can use a protocol called ICE. It consists on a Rendezvous server connected to the Internet. The initiator can advertise the group name on this server (through the Internet). Other devices can then consult this server and discover the group name. When someone connects to this group, a path must be found between the peers in order to allow them to communicate together.

In order to limit the number of groups visible on a network, each application using AllJoyn must declare a so called well-know name. The advertising and discovering of a group is then limited to this well-know name. Only applications using the same well-know name can see each other.

As mentioned earlier, in AllJoyn someone has to host the session. This means that if the host goes down, the session is also destroyed. The initiator can also lock the group so that nobody can join this group anymore. In addition, AllJoyn provides the functionality to detect when a peer joins or leaves the group.

The AllJoyn framework also provides authentication and encryption using various protocols like SRP key exchange, SRP logon and RSA key exchange.

8.2.4. Unicrypt

Within the context of another project, the e-voting research group of BFH developed a library containing the most used cryptographic functionalities in e-voting. This library called *Unicrypt* implements encryption schemes, signature schemes, hash schemes, proof schemes, commitment schemes and other cryptographic building blocks from a mathematical point of view. Proof, hashes, signatures, cipher or plain texts are all considered as mathematical elements. This orientation is

[4] In the Internet of Everything, AllJoyn Enables the "Internet of Things Near You": https://www.alljoyn.org/

useful in e-voting because it allows to implement more easily cryptographic protocols that, most of the time, base on a mathematical background. UniCrypt is written in Java. This library has been used to implement the protocol in this project. The current version is still under development, so this project was a test for this library.

9. Implementation

The main part of this project was the implementation of the protocol and the creation of the user interfaces. Moreover, the networking layer had also to be put in place. This chapter describes the architecture that is used in the app and comments the graphical user interfaces, the network layer and the protocol implementation.

As mentioned earlier, a requirement to use this app is that all the voters are present at the same location as this is usually the case for boardrooms. This requirement simplifies some common challenges in e-voting. Indeed, some assumptions can be made out of this requirement. For example, one assumes that the voters can talk together during the whole voting process, so a voter can manifest themselves if something is not clear for them. Another assumption is that a voter can ask their neighbour to show their device in order to check if they have the same content or the same result on their screen. This simplifies considerably the process at least from the point of view of the developer, because they have not to care about that in the implementation.

As this was already mentioned in the introduction, some work has been done as joint work by the two students. This concerns the user interfaces and the network layer. Although there was not a strict task separation, Jürg Ritter worked more on the look and feel, the Wi-Fi connection, NFC feature and QR-code, while Philémon von Bergen worked more on the UI flow and the logic behind, as well as on the communication layer. However, this was no strict task separation.

The resulting application was called *MobiVote* in order to create a relation to *UniVote*, an Internet voting system for universities also developed in the e-voting group of the BFH[1]. MobiVote exists in two variants. The first one is the product of Jürg Ritter's thesis. It called MobiVote CGS97 in reference to the crypto protocol implemented. The variant obtained in the present thesis was called MobiVote HKRS12 also in reference to the authors of the protocol used.

The minimum version of Android required to run *MobiVote* is version 4.0. This limitation is due to the use of some functionalities that are only available since Android 4.0.

9.1. User Interfaces

This project was preceded by a preparation project where different tasks were done. Among other, a first implementation of the protocol was done as proof-of-concept and a thought on the graphical user interfaces was also realized. This chapter first presents the result that was obtained in this preparation project, since it is what has been implemented at the beginning of this project. During the development and after some usability tests, some modifications have been done to this first version. The storyboard showed in chapter 4 is the final storyboard, so it differs from the one that

[1] https://www.univote.ch

is presented in section 9.1.1. However, since some interfaces are similar, some information could be redundant.

9.1.1. Initial User Interfaces

For the design phase of these user interfaces, various people sat together in order to have different opinions and thus trying to obtain the best possible result. Both students implementing the app and their supervisors were present. The goal was to design the UI as usable as possible.

The first thought was to separate the role of vote administrator and simple voter. The administrator has to set up the network configuration allowing the message exchange. They also have to define the vote properties as the question, the proposed options and the electorate. They then can start the voting period and must be able to end it. The simple voter only wants to be able to vote and to see the result. In order to separate these two roles, it was decided to develop two different apps. One, the *Admin App*, is used by the vote administrator and the other, the *Voter App*, is used by the simple voter.

Some functionalities are used in both apps, as for example the voting functionality. The administrator, as well as the simple voter, often has the right to vote. For that, it was decided to create library applications providing these functionalities that could be used by both apps.

The following figures represent the user interfaces of these different app and libraries. Figure 9.1 shows a part of the UI of the administration app. Figure 9.2 represents a part of the voter's application. Figures 9.3 and 9.4 are library applications providing voting functionalities and display result functionalities respectively. The numbered arrows show transitions from a figure to another.

Admin App

When starting the administration app, it shows a welcome screen (A in the figure 9.1) containing three buttons. The first one, *Setup Network* shows view B where the administrator can define which network will be used and where they can configure the group credentials. Once these elements have been defined, view C displays the connection information needed by other participants to join the group. This can be done through a QR-Code or NFC. For the QR-Code functionalities, ZXing[2] library was used.

Back to the view A, the second button allows to define a vote. View D shows a list of already defined votes. It is also possible to create a new vote. When choosing this option or by clicking on one vote in the list, view E is shown. There, it is possible to edit the question and the list of options. Buttons allow to save or cancel changes. Another button allows to start the voting period.

This last option moves on to view F. This screen lists the participants present in the group. The vote administrator can select the participants they want to add or remove from the electorate. When they click on the validation button, the vote is sent to each participant who can check if everything is correct in a review screen (view G). If all the voters agree with the vote content, the voting period can be started by the administrator.

[2]Zebra Crossing ZXing: http://code.google.com/p/zxing/

Two cases must be distinguished at this point: if the administrator is in the electorate, the voting library application will be started and will allow them to vote (arrow no 1). If they are not in the electorate, the app will show view H indicating the progress of the vote. When all the voters have submitted their ballot, or when the administrator decides to end the voting period, the result library app is launched to display the result. The fact that the administrator can end the voting phase can be discussed. If they do it before the time fixed in agreement with the voters, these could ask to cancel the vote. However, when a voter gets out of the network and do not submit a vote, it is important for the administrator to be able to end the period.

Finally, the third button of view A allows to show archived votes. This functionality starts the result library application (arrow no 3).

Voter App

The voter app starts with a welcome screen (letter I in figure 9.2) containing two buttons. The first one allows to join a group. The second one allows to show the archived votes. A click on it starts the result library app (arrow no 3).
If the user decides to join a group, they are redirected to view J where they can set their identifier, choose a Wi-Fi network, and indicate the group password or scan the QR-code visible on the administrator's device.

Once connected, view K is displayed. The voter can see the participants connected to the group and can check who is allowed to vote. When the administrator validates the electorate, a view L is shown where the voter can check the settings of the vote (questions, proposed options, electorate). If something is wrong, they can indicate it orally to the administrator who can change it if necessary. If everybody agrees on the content of the vote, the administrator starts the voting period, what results in displaying the voting library application on the voter's device.

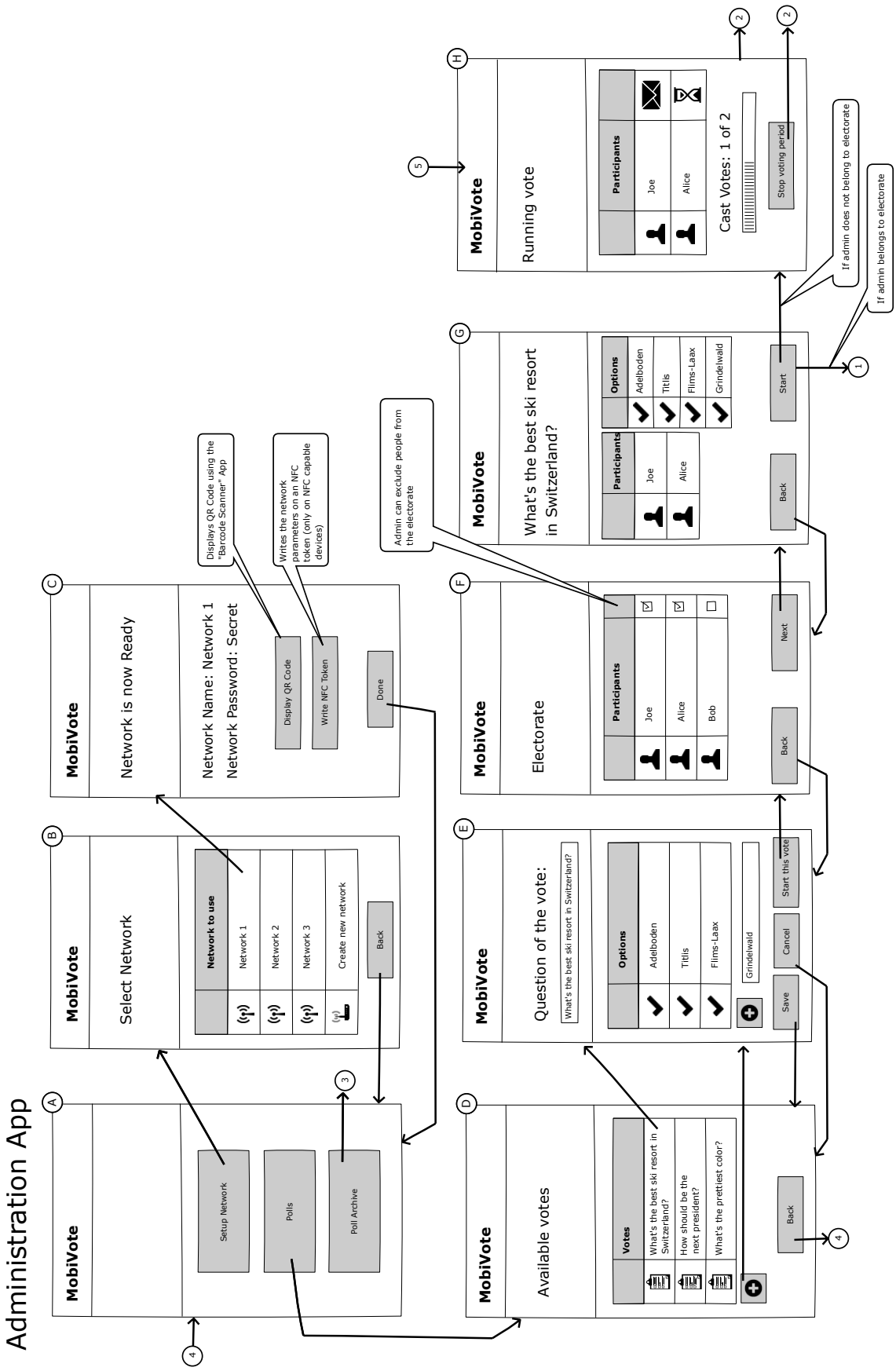

Figure 9.1.: Interfaces of the admin app

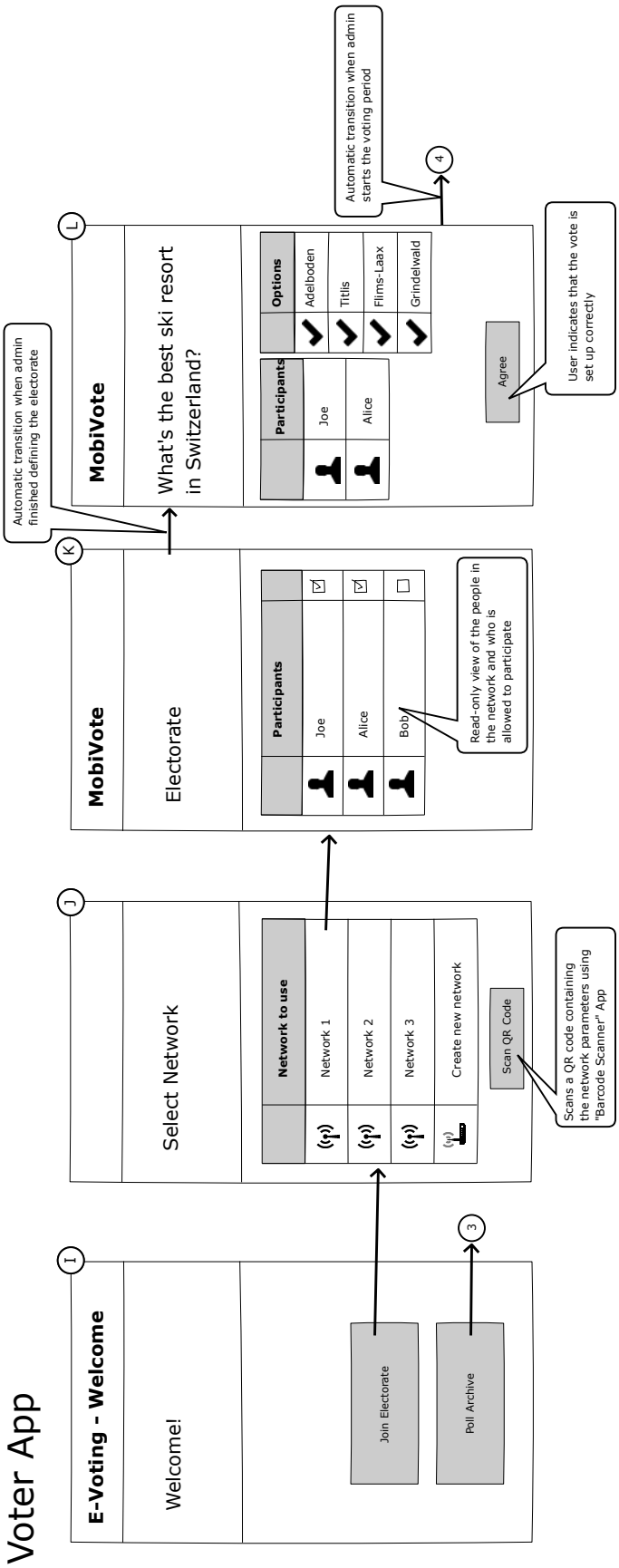

Figure 9.2.: Interfaces of the voter app

Voting library app

The voting library app shows the question and the proposed options (view M in figure 9.3). It allows the voter to choose one of the options. Once done, a progress screen is displayed showing the progress of the vote (view N). When the voting period ends, the result library application is called (arrow no 2).
If the voting library app was called by the admin app, it returns to it as soon as an option has been chosen (arrow no 5).

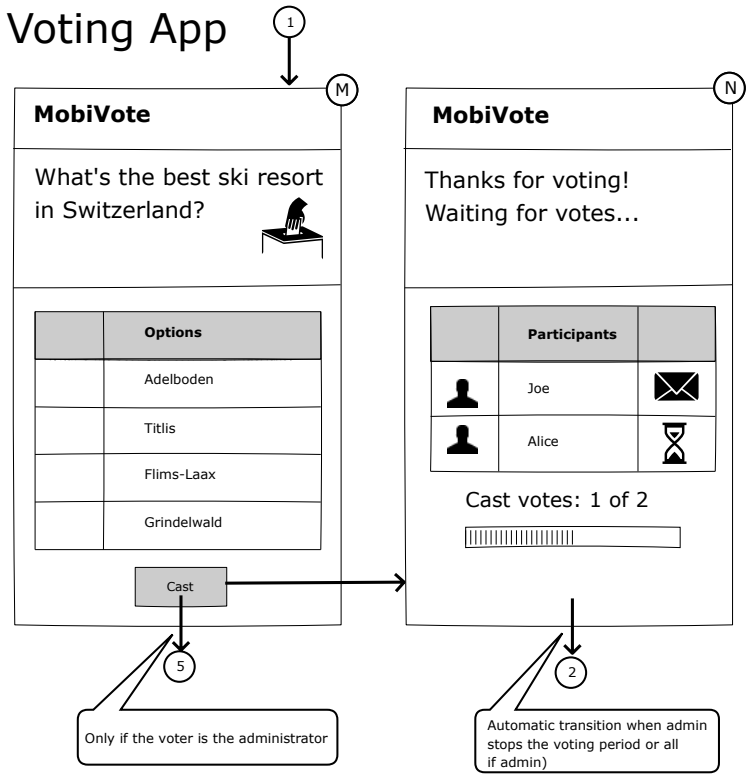

Figure 9.3.: Interfaces of the voting library app

Result Library App

The result library app is composed of two views. The first one (view O in figure 9.4) consists of a list of already completed votes. This view can be accessed from the main screen of the voter app or the admin app (arrow no 3). When selecting an archived vote, it is possible to show its result. The second view (view P) shows the result for the given vote. This view is also used to show the result of a just completed vote (arrow no 2).

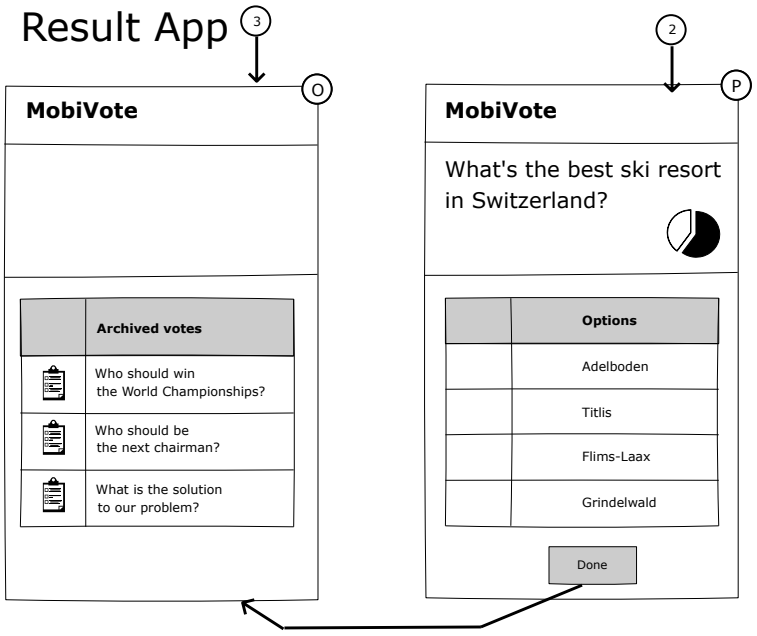

Figure 9.4.: Interfaces of the result library app

This was the storyboard that resulted from the first discussion and that was first implemented in this project.

9.1.2. Modifications Done to Original Flow

During the implementation of the initial storyboard, a major problem appeared. When Android runs an app, it gives it a context. This context is generally the Java package name of the application. This context is used to store application private data. Only the app running with this context can access these data. These data are composed, among other things, of the database of the application. In MobiVote, the goal was to have only one database for the admin app and the voter app, in order to be able to retrieve the archived votes regardless of the fact that they were run as admin or as voter. But the problem was that the voter app and the admin app received a different context when running and thus had both their own database. We (the two students) thought about updating both databases with the content of each other. But this was not satisfactory because of the problems that could result if an app is uninstalled. Asking the user to install both apps was not very usable and could generate problems in case that one of them would be uninstalled.

Similarly, some usability tests proved that having two apps was not a good idea because the user sometimes did not know which app to choose. So it was decided to switch to a single app, offering both functionalities in a more understandable way and thus solving the database problem. Of course, this modification took time and caused a delay compared to the expected planning.

Another significant change was the order of creation of the vote and the set-up of the network communication. In the initial storyboard, it was planned that the user playing the administrator role first sets up the network communication (with the first button on the main screen) and then defines the vote (with the second button on the main screen). But usability tests showed that the user did not care about setting up a network because they did not know the reason to do it and

thus directly did what they wanted to do: setup a new vote. So we decided to change a little the flow of the administration process by first asking the user to define its vote and then ask them to define the network.

For these reasons, it was decided to switch to a standalone app. So, the UI flow had to be adapted. The interfaces themselves did not change a lot, only the order of displaying them was adapted. This new flow had significantly better results in usability tests, even if the interfaces themselves did not change a lot. The result of this revision was already presented in chapter 4 and thus is not repeated here.

9.1.3. Implementation of the Storyboard in Android

The implementation of these storyboards to concrete Android user interfaces raised a lot of questions and implicated some adaptations. This section describes some of them and the solutions that were chosen. This section shows some screen captures of the final application to illustrate the solutions that were implemented. The previously mentioned usability tests also showed a lot of small details that were not satisfactory. They are also discussed here.

Position of the buttons As this was already mentioned, we made a first version and did some usability tests with different people. In this first version, we implemented quite the same look and feel as in the storyboard, so we placed the buttons at the bottom of the screen. We created a bar at the very bottom of the view, where the texts of the buttons appeared. This bottom bar took the entire width of the screen. When we tested it with other people, we noted that those who were using a tablet did not see the button, because it was too wide. So, we had to change that. We analyzed where we could place these buttons and noticed that we had two types of buttons: the flow buttons used to manage the flow of the app (like *next*, *previous*, *start voting period*, etc) and the action buttons (like *save*, *cancel*, *cast*, *agree the review*, etc). We found that action buttons could be replaced in most of the cases by dialogs. For example, *save* or *cancel* buttons in the vote creation screen could be replaced by a dialog asking for saving or cancelling when leaving this screen. The flow buttons however could not be replaced by something else.

As we could remove the action buttons, we needed less space to display only flow buttons, so we decided to put them in action bar on top of the activity, which is Android's standard way to do it. But this was not satisfactory on smartphones, because there was not enough space in the top bar to display the activity title and the flow buttons. Since the bottom bar problem only appeared on tablets, we decided to implement following solution: on smartphones we would display these buttons in a bottom bar since it was not a problem on this type of devices in the usability tests, and on tablets we would display them in the top bar, since we have enough space to show them there (see figure 9.5 and 9.6).

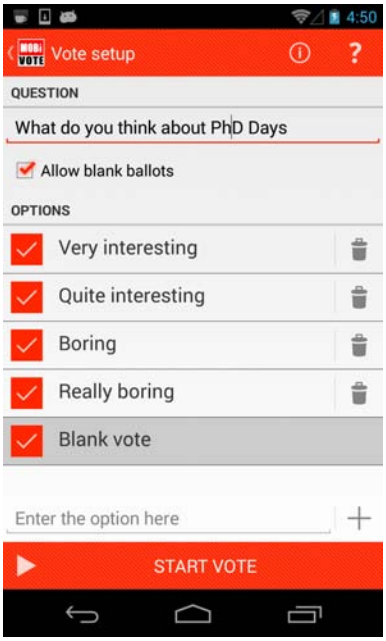

Figure 9.5.: Buttons at bottom on smartphones

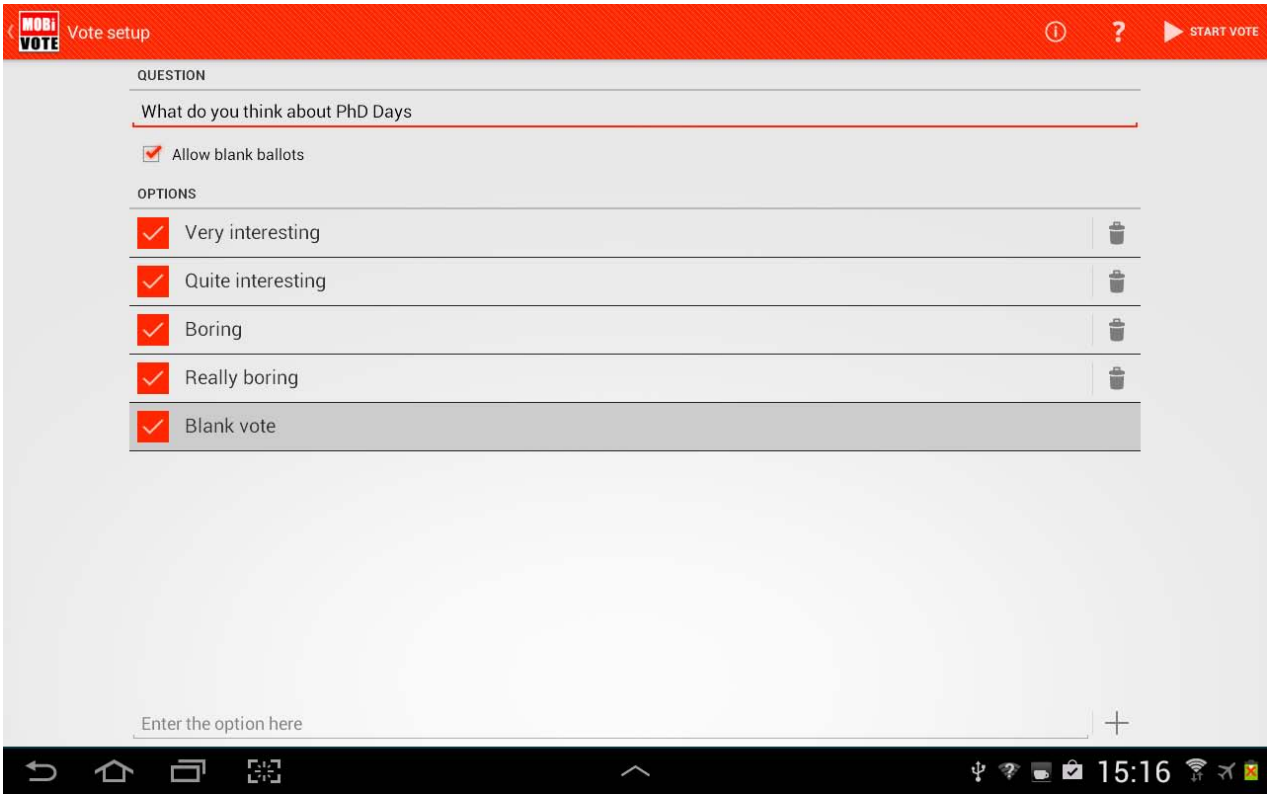

Figure 9.6.: Buttons in top bar on tablets

Special buttons In addition to the buttons we just mentioned, we decided to add two buttons. One is a help button displaying a dialog containing information about the currently displayed activity. The other is a button allowing to display the network information like the network name

and the group credentials. The reason for adding this last button was that the user should be able to show this information to a newly arrived participant even if the administrator has already closed the view containing these data.

We also had to add a button allowing to go back, respectively to go in hierarchical reverse order of the activities. The back button present on most of the Android devices allows to go in timely reverse order. So, there is a flow of views and the user wants to jump over one when going back, they cannot use the standard hardware back button. For this purpose, the standard Android way is to transform the application icon in the top bar in a button allowing to go back to a predefined activity declared as parent activity. This is the way we did it. It is also the reason why the hardware back button and the parent button (the one we just talked about) sometimes does not have the same behavior (see figure 9.7).

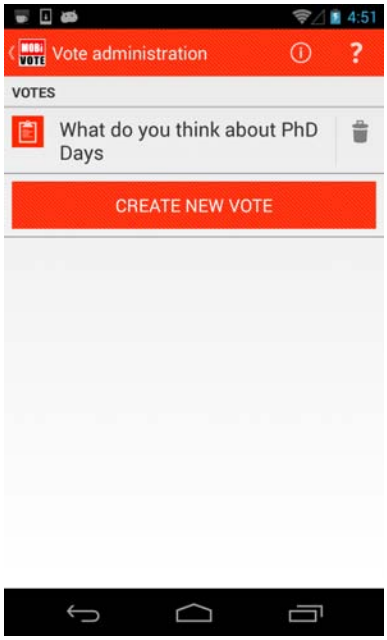

Figure 9.7.: Special buttons in top bar

Tablet layout Smartphones and tablets have different screen sizes. Moreover, a smartphone is mostly used in portrait format since tablets are rather used in landscape format. So, some activities have been designed differently in portrait and in landscape format in order to improve usability and to be as compatible as possible with these types of devices.

Activity specific adaptations Some adaptations had to be made to specific activities. This paragraph describes them.

Vote setup activity In the first study, we thought that the administrator should define all the options they want to allow the voter to choose. But, when using the app, we noticed that it would be helpful to be able to easily add and remove the blank option, as this could be used quite often. So, we added a checkbox that allows to add and remove this option in one click (see figure 9.8).

Another problem that occurred in the usability tests was that people began to edit an option but forgot to add it to the list by clicking on the "+" button. So, we added a check that if an option

is still being edited when starting the vote or leaving this activity, a dialog is shown to ask if the option has to be added to the list.

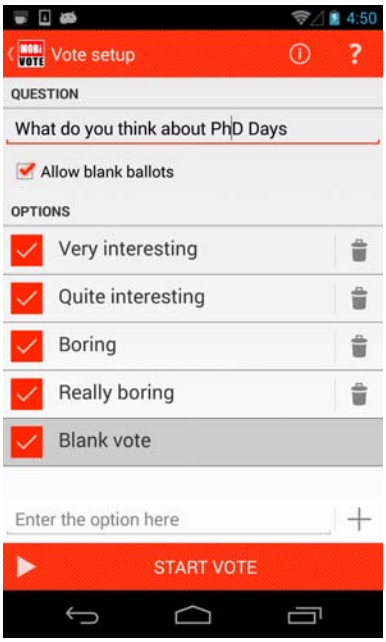

Figure 9.8.: Blank vote and "+" button

Network setup activity In this activity, the user has to specify an identifier (a name) that will appear on the device of the other voters. In a first version, the default value was set by reading the identity information configured in Android. In order to access these data, the app had to request two permissions, namely READ_CONTACT and READ_PROFILE. This means that when somebody wants to install the app, they have to accept that it reads the contacts stored on their device and access to their personal information. These are exactly the permissions that people do not like to give to an app because they do not know what the app will do with these data. For an e-voting application, it can be suspicious to use these permissions. So, we decided to remove the default identification feature in order to be able to remove these permissions. In the final version, we check if the identification field is empty and notify the user if it is.

As mentioned before, a voter has four ways to join a group. The four possibilities were proposed with four buttons (in the same order as described here). The first way is to use the Wi-Fi network that is currently connected and to type in the group credentials. The second one is to scan a QR-code containing these data. The third one is to scan a NFC tag and the last one is to connect to another Wi-Fi network and type in the group credentials (advanced network config). When testing the app, we noticed that the easiest way was to scan a QR-code. So we changed the order of the buttons and put the QR-code and NFC buttons first, followed by the current network button and finally the advanced network configuration button. This seemed to be the most logical order.

The connection to a Wi-Fi network was quite a tricky part. There are lots of cases to handle. The user can connect to an unsecured network, to a secured and already known network (the key is known and saved in the device), to a secured but unknown network or even to a secured known network but for which the key stored in the device is no more up to date. These are a lot of use-cases. In the cases where the network is secured but unknown, a field where the Wi-Fi key

can be indicated must be displayed. In the case that a network is already known but its key was changed since the last connection, the process will fail and the user will have to go to Android Wi-Fi Manager in order to remove the not anymore used Wi-Fi key. All these use-cases had to be managed in case of entering the group credentials manually and in case of scanning a QR-code or reading an NFC tag.

There is one use-case more: if no wireless access point is available, the admin can use the built-in hotspot of their device. So, they must be able to set-up a network indicating a name and a Wi-Fi key.

Review activity The purpose of the review activity is to allow each voter to check if the content of the vote and the electorate are correct. The voter has to accept it so that the administrator can go to next step. First, the accept button was placed in the bottom bar / top bar, but most of the time, people did not see it. So, we moved it in the electorate part, on the line corresponding to the voter. So, the user had to click next to their name. But, because of esthetical reasons, the button was not recognized as such. So, we finally transformed it in a checkbox. With this solution, it was more understandable.

In the help of this screen, we had to indicate that, if a voter, for some reason, does not accept the review, the administrator has to exclude this participant from the electorate in order to run the vote. This information is also displayed when the admin tries to go to next step but not every voter has accepted the review (see figure 9.9).

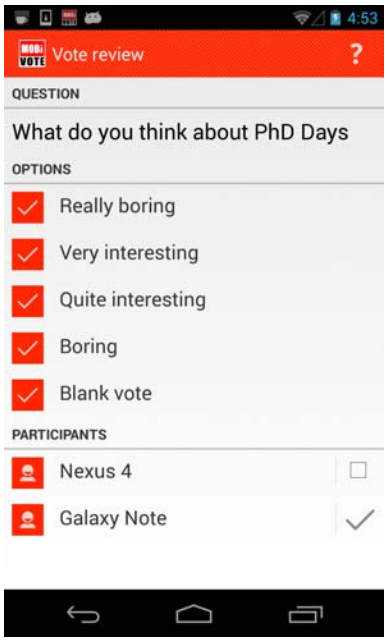

Figure 9.9.: Review screen with checkbox

As this activity and the network configuration activity were not very understandable for the users, we implemented help overlays that are displayed the first time these activities are shown. These overlays contain arrows and small texts indicating what the user has to do in this activity (see figure 9.10).

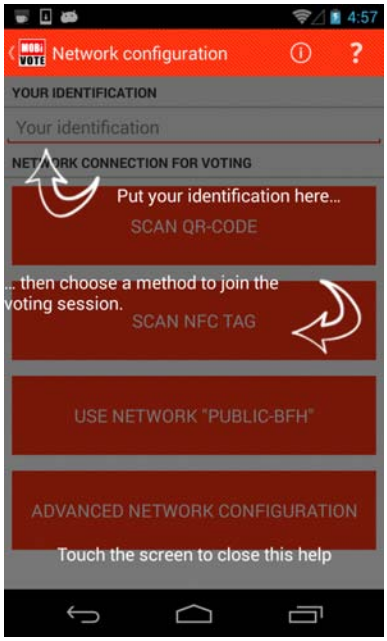

Figure 9.10.: Help overlay

Vote activity When voting, the voter should always have to do it in two steps in order to confirm their choice and not accidentally choose something they did not want. In the first implementation, they had to choose an option with a radio box and to confirm the choice with a button. But since we removed all action buttons, this button was also deleted. So, we had to find another way for the user to confirm their vote. In the final version, a dialog is shown when the user chooses an option, asking them to confirm their vote.

Wait for incoming votes activity In the activity where the admin is waiting that each voter casts their vote, they have the possibility to end the voting period in case a participant does not submit their vote. Another functionality was added allowing the admin to cancel the vote, if something goes wrong.

Result activity In order to improve the readability of the result, we decided to add a pie chart showing the result in form of a picture. Therefore, a library called AChartEngine[3] was used. We also added some information to this screen, namely the number of people in the electorate and the number of ballots submitted. We also decided not to show the members of the electorate in this screen, because this is not a property of the vote itself but depends from the setup. Just after the voting period ends, one still knows who was in the electorate, and if one consults the result afterward, the interesting point is the result and not the electorate.

It can happen that a member of the boardroom comes too late to the meeting and the voting period was already started. In such cases, the administrator can end the voting period, thus showing the result screen. Then they may like to start the vote again. This is possible with the *repeat vote* button. This action only copies the vote and shows the vote editing screen in order to allow the administrator to start it again. This button is only visible by the administrator and just after the voting period ended (see figure 9.11).

[3] http://www.achartengine.org/

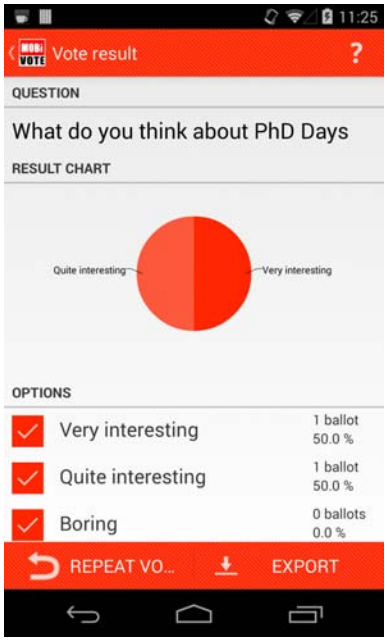

Figure 9.11.: Result screen with pie chart and "repeat vote" button

When accessing to this screen through the archives, a similar button is shown, this time not only to the administrator, allowing to copy the vote, in order to run it with another group. A typical use-case for this feature would be the professor who does a poll with a class of students, and would like to repeat the same poll with another class without having to redefine the entire poll.

Another adaptation is the addition of the *Export* button. The protocol is verifiable, so someone would maybe like to have the possibility to verify, after the voting process, if everything worked correctly. This could be done by an external verification software. So, there must be a possibility in the app, to export the values generated during the protocol run. This button offers this feature. It exports the protocol values in an XML file, which can be used in an external verifier.

General adaptations Some more general adaptations were also done after the first tests. For example, error messages were sometimes shown in toasts (small text appearing at the bottom of the screen) and sometimes in dialogs (overlay window that must be closed with a button). We finally decided to use toasts for informal errors that do not need the vote to be interrupted, like the case of a voter who wants to connect to a group but forgot to indicate their identification. Dialogs are now used for critical errors or information that require the vote to be interrupted such as when an attack is detected on the network. Dialogs are also used for questions, since there must be at least two buttons (yes and no) to answer the question.

Another adaptation of the first version is the title of the activities. Initially, the title of the activity was the name of the application. In the activity, we then displayed a title describing what the user has to do. We however noticed that these titles could be placed in the activity title (as this is an Android standard) and this would offer a gain of space in the activity itself. That is what we did.

9.1.4. Other Features

Other features were also included in the application. For example, there must be a way to detect network or group connection loss to avoid that the user waits for messages they will never receive. There can be various reasons for a connection loss: Wi-Fi out of range, device problem, group no more existing. For all these cases, we had to register elements that can be notified if such an event happens, in order to be able to notify the user that something went wrong.

9.2. Communication Layer

In order to allow devices to exchange messages, there must be a communication layer. For this application, it was decided to use wireless LAN as network layer. But this is not sufficient since devices connected on the same network do not necessarily know each other and thus cannot communicate together. This was the reason for using InstaCircle or AllJoyn. These frameworks allow to create a group of devices knowing each other and make it possible for them to communicate together. They also limit the communication between the members of the group, so that other devices connected on the same network but not member of the group cannot listen to the communication.

The first thought was to use InstaCircle and replace the chat application built over it by the voting application. However, there was a problem with it. For an unknown reason, on some networks, a big percentage of the messages got lost for some of the participants while on other networks it worked reliably. This problem was already discovered in a previous project and a resend mechanism was implemented but it was not satisfactory. However, since we did not have another variant at the beginning of the project, a first partial implementation was done with InstaCircle.

During the first phases of the project, one of the two students came upon the website of AllJoyn. By studying it thoroughly, we found out that it could correspond to our need. So, we decided to make a try with AllJoyn. A first partial implementation could be set up quite easily. It worked much more reliably. Since InstaCircle still had some missing functionalities as did AllJoyn, we decided to implement them on the most reliable one: AllJoyn. However, some functionalities from InstaCircle were kept, like the WLAN management or the logic of transmitting the network group credentials.

The functioning of AllJoyn was already described in the specifications section, but we repeat it here since it is important for the understanding of this section. AllJoyn works as follows: someone, let us call them the initiator, has to create a group and give a name to this group. The initiator's device will host the group. The group name is then advertised on the network through multicast. Another device using AllJoyn can discover the groups available on the network. If there is one, the device can connect to this group. The communication between the devices connected to the group takes place over unicast transiting over the initiator hosting the group. A message sent to the entire group first goes to the initiator which transmits it to each participant of the group. In order to limit the number of groups visible on a network, each application using AllJoyn must declare a so called well-known name. The advertising and discovering of a group is then limited to this well-known name. Only applications using the same well-known name can see each other.

As seen before, the initiator must define a group name. The only restriction is that it must start with a letter. So, the initiator could define a group name related to the vote. This can be helpful

in order to identify the group more easily, but this implies that the people wanting to join the group have to type the group name in their device. Moreover, this reveals some information on the vote over the whole network. So, in order to increase usability of the app, we decided to automatically create a group name of the type "group" followed by a number, where this number starts at 1 and is incremented for each new group created on the same network at the same time. So, this solution allows people that want to connect to this group to only type in the number of the group, and the rest is added automatically. This is more practical for the user and more secure since an attacker does not know what is done in this group.

The implementation of this network layer was based on an example provided with the AllJoyn framework. By reusing this code, it was quite easy to implement a working product. However, there were missing functionalities like all the security aspects and the support of participants' well-known names. These features had to be implemented on top of the example code.

9.2.1. Security Assumptions

As one can see in the functioning of AllJoyn, the communication is no more completely public since it is done by unicast. The protocol however specifies that the discussion must be public in order to satisfy the *dispute-freeness* property. In order to solve this problem, one has to assume that AllJoyn is trustworthy. One has to trust that when somebody wants to send a message to all group members through AllJoyn, AllJoyn really transmits the same message to each group member. Such a trust assumption can be done relatively easily since AllJoyn is a widely used framework for peer-to-peer communication.

Another assumption that must be done that applies as much for the network layer as for the protocol and the entire app is that there is no malware installed on the device. If there is one, one has to assume that it has total control over the device and that it can do what it wants, from tricking the user by showing modified user interfaces to interacting in the cryptography. The adopted threat model however considers this as out of scope.

9.2.2. Possible Attacks

The network layer is a sensible level since different attacks could be achieved, like denial of service attacks or injection attacks. This section describes some attacks that could be carried out and presents the solutions implemented to avoid them.

As it was described in the functioning of AllJoyn, the group name is publicly advertised, so everybody that is connected to the WLAN on which the group was created and knowing the well-known name used by the app is able to connect to this group. This allows an attacker to easily connect to a group and participate to the communication, injecting whatever they want. In order to avoid that, it was decided to encrypt the messages exchanged in a group. For that, symmetric encryption was used. Therefore, a key known by all official participants was needed. This is done in form of a password. The initiator of the group generates a password and transmits it through another channel (visually or orally for example) to all other people that are allowed to connect to the group. This can be done thanks to the fact that, in boardroom voting, all the voters are present at the same physical location. So, they can communicate together on a different channel than over the network. From this password, a symmetric key is derived for the encryption and decryption of

the transmitted messages. An attacker should not have access to the password. This would not prevent them from connecting to the group, but from listening and injecting messages since not decipherable messages are simply ignored by the participants.

The password mentioned here must be defined by the administrator. In order to make it more usable, it was decided to generate it automatically. This also allows to ensure a certain security level. However, it must be kept in mind that people that want to connect to the group must type in the password. Of course, when the user scans the QR-code or reads an NFC tag, they do not have to type it in. But this chapter focuses on the fact that it can happen that a user has to type it in. So, for usability reasons, it was decided to only use lowercase letters in order to avoid that the user must switch the keyboard many times (number, uppercase, lowercase, special chars are not visible on the same keyboard on a smartphone). This also avoids confusion of some chars (uppercase o and number 0). In order to have a correct security level, it was decided to use a length of 10 chars, which corresponds to a security of 47 bits. Today, the minimum strength requirement to avoid brute force attacks is 80 bits. However, since a vote lasts maximum one hour, the brute force attack time is limited. So, for this application 47 bits are sufficient.

An attacker that really wants to integrate the group could however try to built a rainbow table and find the symmetric key derived from the password. This problem can be solved in various ways. The first one would be to declare a static salt in the code. This would force the attacker to compute a rainbow table only for the present application. In this case, a high number of iterations in the key derivation process should be used in order to slow down the creation of the rainbow table. But this is not the best solution since it does not completely avoid the creation of such a table. Moreover, the derivation of the key in MobiVote would take much time and thus limit the usability of the app. Another solution is to use a dynamic salt different for each voting session. This would require that the attacker computes a rainbow table for one session. Since such a session does not last more than one hour, it is not possible to compute such a table in such little time. Even more, this allows to reduce the number of iterations needed in the key derivation process, thus accelerating the application. This is the solution that was implemented, since it is secure and fast.

This solution however requires the transmission of the salt to the participants that want to connect to the group. A salt is too big to be typed in, so it must be transmitted in another way. A salt is not a secret so it can be transmitted over the network. That is the variant that was chosen. When a new participant joins the group, the initiator sends the salt. This message cannot be encrypted since the salt is needed to the new participant to compute the decryption key. Since everybody can connect to the group, an attacker could try to send a wrong salt before the initiator is sending the correct one. This would result in a denial of service attack because the participants would encrypt their messages with different keys derived with different salts.

A first thought to solve this problem was to encrypt the salt with the password and a hard coded salt. So, an attacker who does not know the password could not send a wrong salt. However, since the salt is hard coded and thus publicly known, the attacker could build a rainbow table for this first message. When receiving the encrypted dynamic salt, they could use their rainbow table to decrypt it. So, they would find the password and could decrypt the dynamic salt. Thus, they would be able to encrypt and decrypt messages and participate to the group discussion. So, this solution is bad and cannot be used.

Another solution is to transmit some chars identifying the correct salt to the new participant over a secondary channel. So, when they receive a salt, they can check if it is the correct one. This

secondary channel is already used to transmit the password, so it would be possible to add some chars to this password. The user would only have to type some more chars. The easiest variant would be to take the 3 first or last chars of the base 64 encoded salt and add them to the password. However, if the attacker has an accomplice in the group, the accomplice can reveal these three chars to the attacker who just has to find a new wrong salt starting or ending with these chars, and the DoS attack is successful. So, using the 3 first or last base 64 encoded salt is a bad idea. A variant of this idea is to use some chars from the *hash* of the salt to identify the salt. Reproducing the attack just described before would mean to the attacker that they have to find a salt whose hash has these chars, which correspond to find a partial collision of hashes. The probability of finding such a collision is assumed to be small enough to avoid an online attack.

With the solutions presented above, the problem of enclosing the group and allow access to only allowed people was solved. But someone in the group could still try to impersonate another member of the group, what means sending a message saying that it comes from another sender. In order to avoid these types of attacks, each participant should sign the message they sends with their private key so that nobody else can impersonate them. This however requires a public key infrastructure, what is not really easy to set-up in a decentralized way without third trusted party. So, it was decided to do it in a simpler way. Every participant that joins a group computes a private and public key pair and sends a message containing their public key to all other participants[4]. The participant then signs all the messages they send (including the one containing the public key) with their private key. All the other participants store the public key received from this participant and check the signature of the messages sent by them with this key. Now, the attacker could try to send a public key for a new participant before the new participant had time to do it themselves, so the attacker would be able to impersonate the new participant. This problem is solved by listening if multiple public keys are sent for a participant. If two or more different keys are sent for the same participant, it means that someone is trying to impersonate this participant, so the voting session can be interrupted.

The solution of computing an own key pair and sending the private key allows to create a new pair for each session, so the private key must not be saved, what would require asking the user for a password to encrypt it. Since a voting session does not last a long time, a big key pair is not needed. 512 bits keys are sufficient. AllJoyn also provides an authentication mechanism that was not used here for various reasons. First of all, it is quite badly documented, so a lot of things are not known precisely. Moreover, it seems that encryption and authentication are indissociable, while only authentication and integrity are needed. Finally, AllJoyn's mechanism does not allow to create what is really needed (for example small keys are sufficient for the present use case). Because of these reasons, it was decided to implement it ourself, in order to adapt it to these specific needs.

9.3. Architecture of the Application

The application consists of three main components. The first component manages the flow of the graphical user interfaces. The second component is the protocol implementation. Finally, the

[4] This message is anyway needed to transmit the well-known name of the participant. AllJoyn uses random strings to identify group members, but for MobiVote, a well-known name is required. So, this message is used to transmit this well-known name as well as the public key corresponding to this participant.

third one is the network component, which cares about the message exchange.

The application works in two phases: first the vote preparation phase and second the voting phase. In the first phase, the GUI component interacts with the network layer. At this time, the protocol is not relevant and is not needed. In the second phase, the GUI interacts with the protocol and the protocol interacts with the network layer. Figure 9.12 shows how the components are organized. The red connections stand for the vote preparation phase and the blue ones for the voting phase.

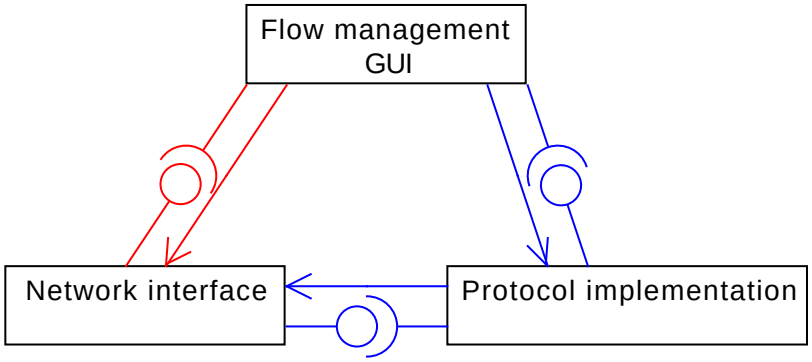

Figure 9.12.: Organization of the components

In this figure, one can also see that in one way a direct reference is used from the flow manager to the protocol implementation and to the network interface. In the reverse way, the concept of events is used to notify the other components about a change. In the implementation, this is done using the LocalBroadcast from the Android system.

Flow manager component The flow manager consists essentially of Android classes like Activities, Fragments and other GUI concepts. It defines the flow of the user interfaces and interacts with the user of the application. It also defines a data model for the votes. This data model contains three main classes as shown in figure 9.13.

Poll is the class representing the vote. It contains the question, a list of *Option* objects which represent the possible responses that can be chosen and a list of *Participant* objects that represent the voters. It also stores the number of participants at the beginning of the vote run, the time when the voting period was started and if the vote was already run or not. It also contains an id serving to identify the object.The *Option* class contains the properties of an option, like the option's text, the number of votes an option received and the corresponding vote percentage. It also contains an identifier and a reference to the poll object by storing its id. The *Participant* class contains the properties of a voter, like a unique identifier on the network, a well-known name and different information flags indicating, for example, if the voter has already voted or not. So, a *Poll* object contains all the information of a vote and can be transmitted from an activity to the other.

The flow manager component is also responsible for maintaining the database storing the votes. The content of this database is a mapping from the data model described before in SQL entries. So, the *Poll* object is stored with its properties, among other, its list of *Option* objects. The *Participant*

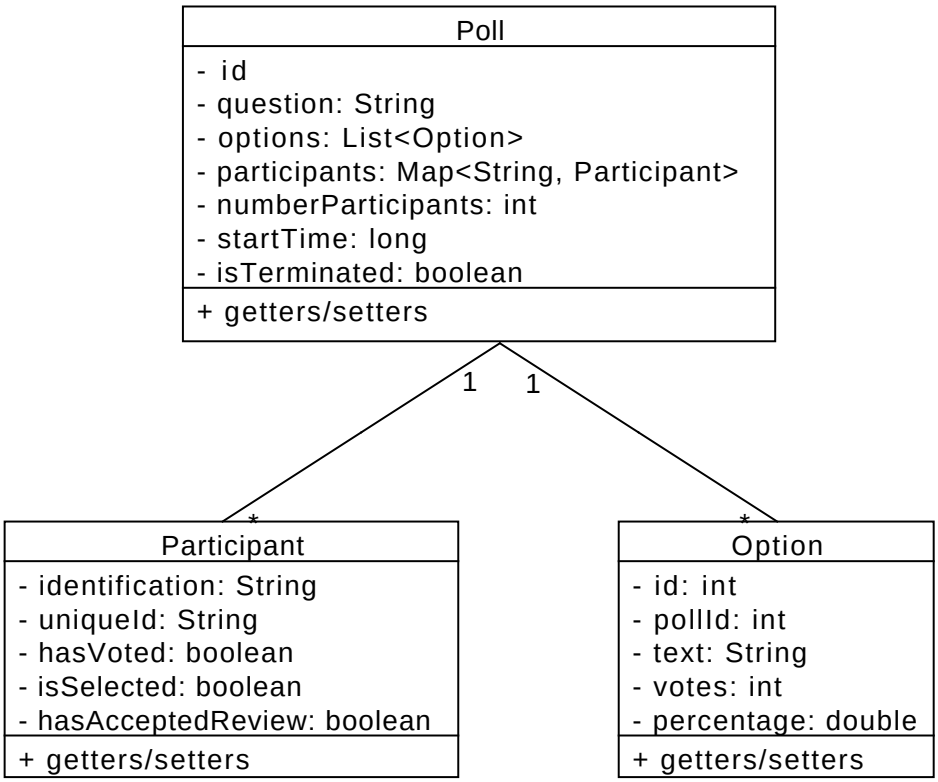

Figure 9.13.: Data model for the vote in the flow manager component

objects however are not stored in the database, because they are considered as dependent of the setup from the network. Thus, it make no sense to store them, because they could not be reused later on another network.

Network component The network component consists of the wireless network management classes and of an interface to the network layer. The implementation of the network layer itself is relocated in an Android library project[5] which is referenced in the main project. This interface and the Android library concept allow to easily change the network layer used which is chosen through the strategy pattern. This was a very useful feature for the change from InstaCircle network layer implementation to AllJoyn.

To exchange messages between the devices, the network layer needs to serialize the objects that must be sent, since only char sequences can be transmitted over the network. For this implementation, Java serialization was chosen since it is a ready to use serialization method. However, this solution is not the best in term of compatibility with non-Android devices. So, a strategy pattern was implemented which allows to change the serialization method without much effort as shown in figure 9.14.

[5] An Android library project is an Android project which cannot be run alone on a device. It needs to be referenced and used as a library by an Android application project.

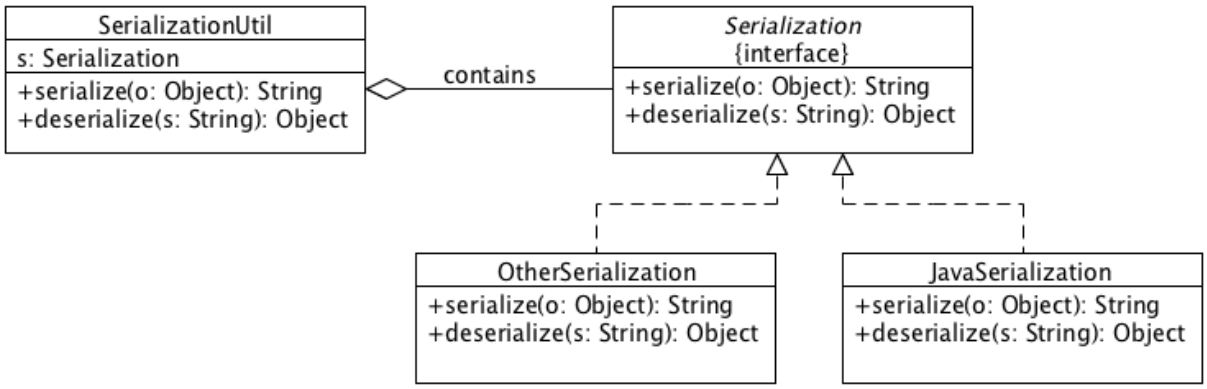

Figure 9.14.: Strategy pattern for serialization

The class *SerializationUtil* is the context class. It references an object of the abstract type *Serialization* which is of the concrete *JavaSerialization* or *OtherSerialization* type. A call of a method on the object *SerializationUtil* calls the corresponding method of the object s. So, the behavior of the methods of the object *SerializationUtil* can be changed easily (even at runtime, but it is of less interest for this case).

Protocol component The last component is the protocol implementation. The flow manager component was designed to fit for different protocols. So it was important to easily be able to change the protocol component. Implementing the strategy pattern for the protocol interface in the flow manager component was a good solution to have a component compatible with these various protocols.

As the protocol description in chapter 3 shows it, the flow of the protocol is very sequential. So, a good way to implement it was to use a state machine. Figure 9.15 shows the flow of this state machine.

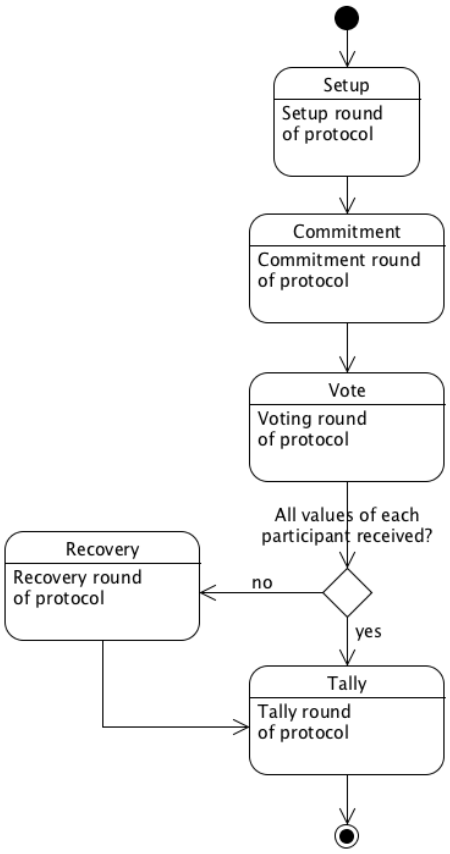

Figure 9.15.: Flow of the state machine

For this purpose, a library providing a state machine implementation[6] was used. Figure 9.16 shows how it works.

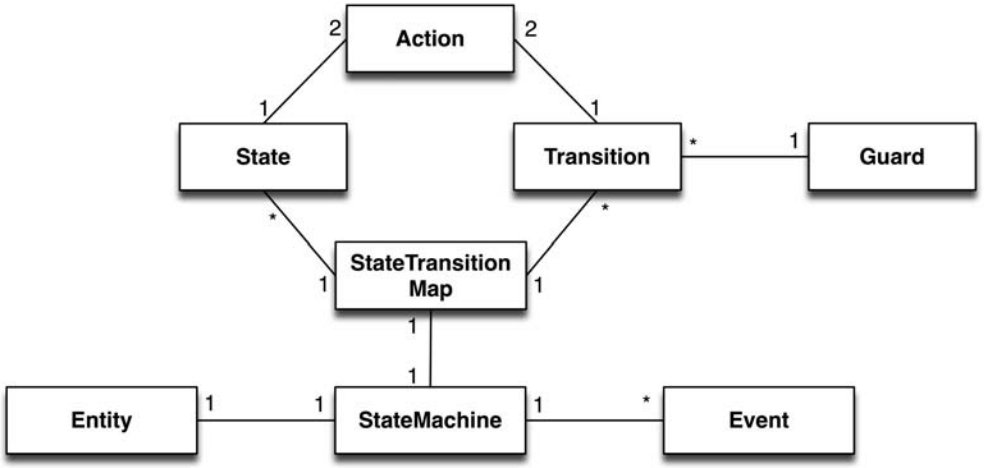

Figure 9.16.: Entities of the state machine

[6]Tungsten: http://sourceforge.net/apps/mediawiki/tungsten/index.php

The main class is called *StateMachine*. It is the class that manages the state of the state machine. This class has a reference to an *Entity*. This entity can be used everywhere in the state machine. So, it can be used to reference an object that would be used in the different states of the machine. The *Event* class represents the events sent to the state machine. For each new incoming event, the state machine looks for a transition corresponding to the received event. If one applies to the current state, the transition is executed and the state machine enters a new state. The transition is defined with a *Guard* which is used to check if an event corresponds to this transition. The *State Transition Map* allows to define the states of the machine and the allowed transitions. This map is passed to the state machine when it is started. Each state and each transition can contain an *Action* which defines what must be done in a state or during a transition.

In this project, only actions in the states were used. These actions contain the implementation of the cryptography of the different rounds of the protocol. Each action listens for incoming messages from other participants, each action processes these messages, some of the actions have to send a message, and so on. One can see that the structure is quite similar and some functionalities are required by all actions. So, an *AbstractAction* from which all actions inherit was created. Figure 9.17 shows its structure.

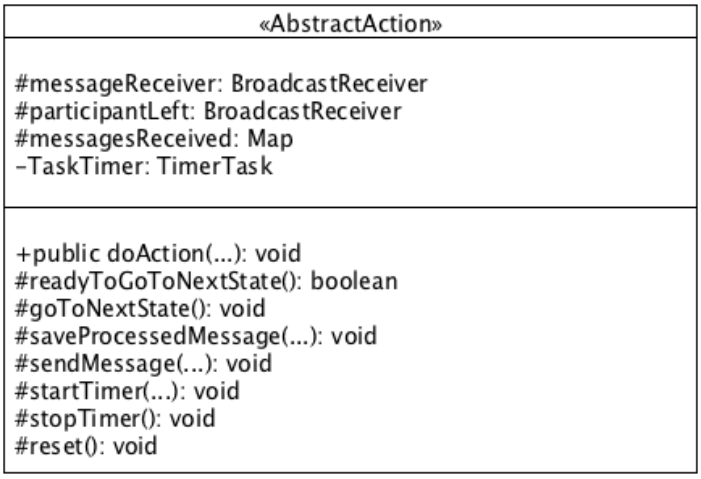

Figure 9.17.: Structure of an action

The *messageReceiver* broadcast receiver is responsible to listen to received messages corresponding to the current round of the protocol. It stores them in the *messagesReceived* map and sends them to a processing service. When the processing is done, the service notifies the action through the *saveProcessedMessage* method. The *participantLeft* broadcast receiver listens if participants went out of the network and exclude them.

The *doAction* method computes the cryptographic stuff needed at this state and sends the message through the *sendMessage* method. After each incoming or outgoing message, the method *readyToGoToNextState* is called to verify if all messages from all other participants were already received. If it is the case, the method *goToNextState* is called which is responsible to send an event to the state machine.

There is also a timeout for some actions. If a participant does not send a message during a predefined amount of time, they are considered as not available anymore and, thus, are excluded

from the protocol. Not doing this would block the vote if a participant refuses to vote or if they go out of the network.

The processing of a incoming message can take some time. So, it is important to do it on a background thread. During the processing, other messages could come in, so there must be a storing mechanism to save them until they can be processed. For that purpose, an IntentService, which is a standard construct from Android, was used. It implements a queue where the message can be stocked before being processed. When the processing thread is free it takes the message at the top of the list and processes it on a background thread, and notifies the action back that the message was processed and that there are some values to save. This service is called *ProcessingService*.

9.4. Implementation of the Protocol

The first main part of this thesis was the design and creation of the user interfaces. The second one was the implementation of the cryptographic protocol. Some points had to be fixed before beginning that part. First of all, it was decided not to support remote participants. That means that only people present at the same location can participate to the vote. There is no way to vote from another location through the Internet. This choice made the security concept easier. For example, being at the same location allows the participants to double check their screen in order to verify if the content is the same. This would not be possible if someone participated to the vote over the Internet. A second reason for not allowing this was the use of AllJoyn as network layer. AllJoyn is a proximity based network application, so the use-case of vote over the Internet is not supported.

Another point of discussion was the possibility to recover after a network failure. The idea is to allow participants to come back in the voting session after falling off the network (for example if their device powers off or faces network problems). Such a feature implies that all messages exchanged over the network must be saved in order to be recovered if it is needed. Another point is that the protocol implemented here is very sequential and needs the interaction of each participant at the different steps. This means that if a participant misses a step, they cannot participate anymore to the subsequent steps. For this reason it was decided not to implement this feature.

For the implementation of the protocol, the data model described earlier in figure 9.13 had to be adapted to the protocol. The one described previously was thought as a generic data model that should be compatible with almost all protocols that could be implemented. But, for the protocol itself a specialization of the data model was required since every protocol is different. There were different solutions to do it. A possibility was to create a new data model in which the classes contain a reference to the corresponding class of the flow manager data model. This solution however required a data model translation at each interaction between the flow manager component and the protocol component. To avoid that, a better solution was to work with inheritance. Figure 9.18 shows how it works.

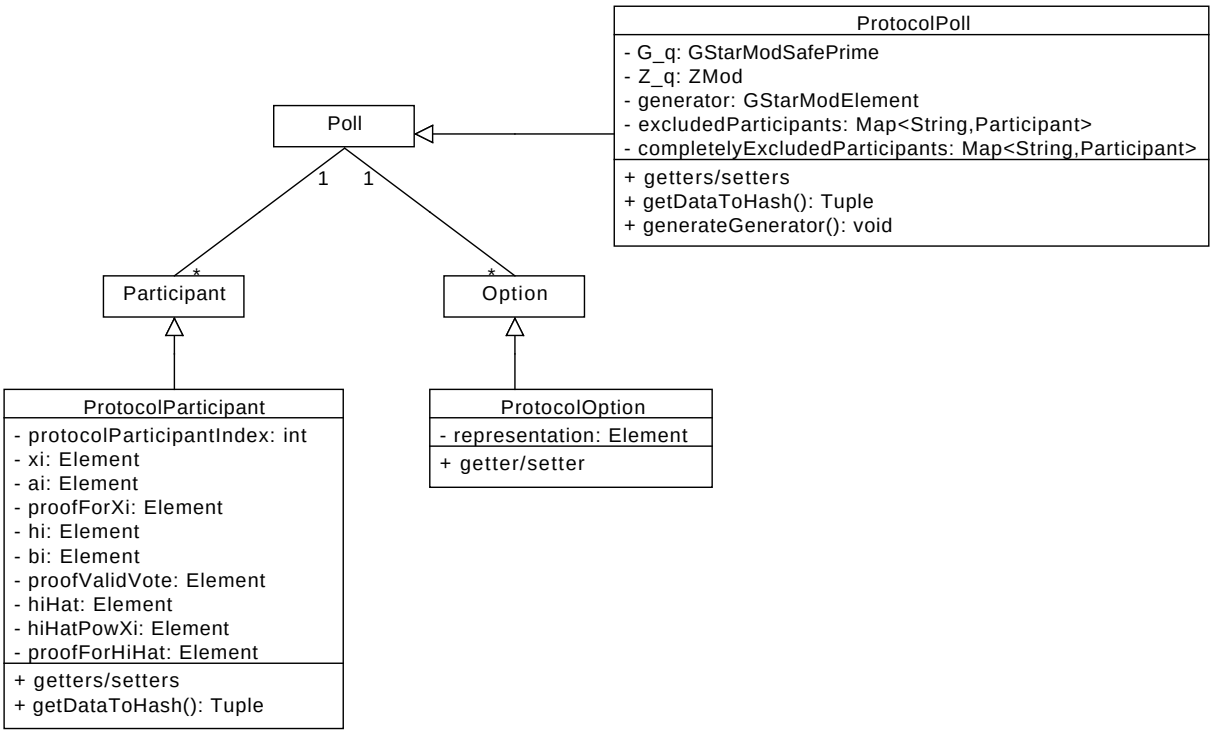

Figure 9.18.: Data model for the vote in the protocol component

With this solution, at the start of the protocol, the flow manager data model has to be converted to a protocol data model. Then, no more translation is needed when sending the protocol data model to the flow manager because the *ProtocolPoll* is a *Poll* and the *ProtocolOption* and *ProtocolParticipant* objects it contains are *Option*, respectively *Participant*. This is the big advantage of this solution. However, it also has a disadvantage, namely that the protocol data model depends on the flow manager data model. But when comparing them, the advantage seemed greater.

The *ProtocolPoll* extends the *Poll* class by defining the mathematical groups and generator (UniCrypt classes) used in the protocol and by storing the participants excluded during the protocol run. The *ProtocolOption* only defines a representation which is the value that is used to encode a vote for this option. Finally, the *ProtocolParticipant* contains all the cryptographic values that are computed at the different rounds of the protocol and it also contains an index identifying the participant in the protocol. *ProtocolPoll* and *ProtocolParticipant* also provide a convenience method allowing to retrieve all the important values that should be hashed in the proofs computed required by the protocol in order to link a proof to a protocol run and to a participant. In the *ProtocolPoll* class, there is also another method used to create a vote dependent generator as this will be explained later.

9.4.1. Protocol Initializations

The architecture adopted for the protocol component is very sequential as the protocol itself is. This was already described. However, the protocol does not start with the setup round. There are some initializations that must be done before. For example, a prime number must be used to

create the mathematical groups, each option must be represented with a value, each participant must be identified. These initializations were described in more detail in section 3.2.1. These are all things that must be done before starting the message exchange required by the protocol. Moreover, each participant must agree on these initializations. That means that they must be communicated to the other participants.

So, the first question was when to do it. Two steps were possible for that: first, just before beginning the review process, second, just before beginning the voting period. The most logical solution seems to be just before beginning the voting period, because it is the moment since which no property of the vote will change anymore. However, a deeper look into this question shows that this solution is not secure. The easiest way to do the initializations is that one of the participants (the administrator of the vote for example) computes them and communicates them to the other participants. Then, these participants should be able to accept the initializations or not. If they are sent at the beginning of the voting period, it is impossible for a participant to reject them. So, it should be included in the review process. The review screen, as described in the user interfaces section, allows the participants of the vote to check if the properties of the vote are correct, and to verify that everybody has received the same vote content by double checking their screen. When including the initializations in the review process, there is a possibility for a participant to reject them, and after the review process, nothing can be modified anymore.

However, there is still a problem. How should a participant agree on the cryptographic initializations? Cryptographic values cannot easily be displayed and verified on a screen. To make this verification possible, there must be an implicit check that does not require the intervention of the voter. The idea was to bind the initializations mentioned earlier with something visible, i.e. the text of the options and the question of the vote. Concretely, the administrator computes these initializations, combines them together in order to obtain a reference string and finally, gets a generator of the cyclic group needed for the protocol using the reference string. The admin then sends these initializations to all other participants which do exactly the same computation with the received values and compare the result with the received generator. If they do not match, it means that the values received were not the same as the ones of the admin. The admin could still try to make an attack by sending different contents to the various participants, but in this case, the generator used in the protocol would not be the same for all participants and the protocol would fail.
This idea has another advantage. In this protocol, there is no central bulletin board or trusted party, so nobody can guarantee that a computed result corresponds to given vote. By using the texts of the vote in the crypto protocol, both parts cannot be separated anymore. So, this guarantee can be obtained without trusted party. Moreover, using the initializations in the choice of the generator prevents the administrator from choosing the generator they want.

The app developed in this thesis supports multiple options vote, not only yes-no votes. Therefore, a way to represent each option in the protocol had to be implemented. The technique of Baudron et al described in 3.2.8 was chosen. This is done in the initialization phase by the administrator, i.e just before the review process. For the creation of the modular cyclic group used in the protocol, a safe prime of 2048 bits is used what is commonly estimated to be secure until 2030.

9.4.2. Setup Round

Once this done, when the admin decides to start the voting period, the state machine described in section 9.3 can be started, and the action defined for each step can be executed. The first one is the *Setup round*. A message is sent at this step containing the value to publish and the proof as described in section 3.2.2. When receiving the message from another participant, the proof is verified and the values contained in the message are stored. If the verification of the proof fails, the owner of the proof is excluded from the protocol.

For this round, a time out is set. If some participants do not send their message in the given time, they are excluded from the protocol. They will not be recovered in the *Recovery round* since this round is foreseen to recover only voters that have participated at least to one round, what is not the case here.

9.4.3. Commitment Round

Once the setup message received from each participant, the state machine changes to the commitment state. This round is cut in two parts. First, a computation has to be done with the values received in the setup round. Once this done, the user has to choose the option they want to vote. Then, the second part of the round is to encrypt the chosen option and compute the proof of validity. At this step, only the validity proof is sent to all other participants.

More detailed, the value computed in the first part of this round is the h_i value as described in section 3.2.3. This value is not foreseen to be published in the protocol description. However, it is needed by the other participants in order to verify the proof of validity received. The verifier of this proof could compute the h_i value as the prover did to generate the proof, but that means that each participant should again compute the h_i value for all other participants, and that takes some time. In order to save this time, this value is sent with the vote which is published in the voting round. Doing this is not a risk for the security of the protocol because the h_i value is not secret since it can be computed by all other participants. The only risk in doing this would be that someone voluntarily publishes a wrong h_i value. But even that would not be dangerous since this would cause the verification of the proof of validity to fail and thus the participant would be excluded from the protocol. So, one can see that publishing the h_i value is not a problem.

As seen in the setup round, a timeout was implemented in order to avoid a deadlock if a participant falls out of the network. In the commitment round, this feature is not used, because using a timeout would limit the time the voter has to choose the option they want to vote. Using a timeout would require someone to define how long it should be (since the duration of the voting period should be configurable) and it would require to show the resting time on the screen of the device. Another problem using a timeout is the problem of synchronization. The timer is started at the beginning of the round, but the device of each participant does not necessarily start the round at the exact same time. There can be a delay caused by the difference of computation power for example. This is not really a problem for the timeout in the setup round since it is only used when a problem occurs (network connection loss of one participant). In the commitment round, the timeout would have another function, namely to indicate the resting voting period time. A voter could want to wait the five last seconds before voting. If there is no synchronization between the devices, it could happen that one device has already finished the voting period and does not accept new votes anymore. Since implementing synchronization between the devices is quite hard, it was decided

not to use a timeout. It is not absolutely necessary since the admin of the vote has the possibility to end the voting period. So, if one device fell out of the network, the admin can play the role of the timeout.

9.4.4. Voting Round

Once all commitment messages received, the state machine changes to the voting step. There, a message containing the vote is broadcast. When receiving the message from another participant, the proof of validity is verified and the values contained in the message are stored. If the verification of the proof fails, the owner of the proof is excluded from the protocol. This step corresponds to the one described in 3.2.4. At this step, a timeout is used in the same way as described for the setup round.

9.4.5. Recovery Round

The recovery round is the last round where a message is broadcast. The computations done in this round are described in 3.2.6. The goal of this step is to make some computations that allow to compute the result of the vote even if some participants did not send a value for some of the rounds described before. The computation done at this step can be studied in 3.2.6. In this round, some messages are exchanged as in the other rounds. To avoid a deadlock if a participant does not send a message, a timeout is used in the same way as described before, in the setup round.

To verify the proof computed in this round, a value is needed that is not foreseen to be published in the original protocol description. It is an identical case as described in the commitment round. For the same reason, this value is published.

In certain circumstances, the recovery round can create some problems with the ballot secrecy. Let us imagine there are four participants at the beginning of the voting period, but two get out of the network. The recovery round will enable the two resting participants to compute the result of the vote. If there are only two voters, there is no ballot secrecy since each voter can find out what the other voted. This means that the recovery round can put the ballot secrecy property in danger. It however was decided not to care about this case, because the ballot secrecy property is not only broken in case of two voters but also when all minus one voters vote for the same option. In this case, the only voter that does not choose this option breaks the ballot secrecy. So, it seemed not appropriate to only block some of all these cases and blocking all (even assuming there is no collusion between voters) would be too limiting.

9.4.6. Tally

The last step of the protocol is the tally. As it stands in the protocol description in 3.2.5, the tally uses a homomorphic process to recover the result of the vote. In the mentioned section, one also observes that there is a discrete logarithm that must be computed. As this was mentioned in section 5.2, the best solution to compute it is to try all solutions corresponding to a valid vote.

Standard method So, an algorithm generating all valid options permutations depending on the number of options and participants was implemented. For all these permutations, it computes the result that should be obtained by the protocol by encoding the permutation as a vote v and by computing the corresponding g^v modular exponentiation. It then compares this value to the one obtained by the protocol until they match. So, one modular exponentiation is needed for each permutation. The time complexity of this algorithm is limited by the number of valid permutation for a specific setup.

Let us have a look at an example. We have a vote with 4 options and 6 participants. A vote for option one can be represented as follows: $[1, 0, 0, 0]$, a vote for option 2 as follows: $[0, 1, 0, 0]$, a vote for option 3 as follows: $[0, 0, 1, 0]$, and so on. The sum of the votes must be equal to 6, since there are 6 voters and every voter has to submit a vote. So, $[3, 1, 2, 0]$ would be a valid result with 3 votes for option 1, 1 for option 2, 2 for option 3 and 0 for option 4. However, $[2, 3, 1, 4]$ is not a valid result for this setup since there are more than six votes in total. For this setup, there would be 84 valid options permutations.

By studying the complexity of this algorithm, it was noticed that the number of valid permutations in function of the number of options o and the number of participants p were contained in Pascal's triangle, as shows figure 9.19[7].

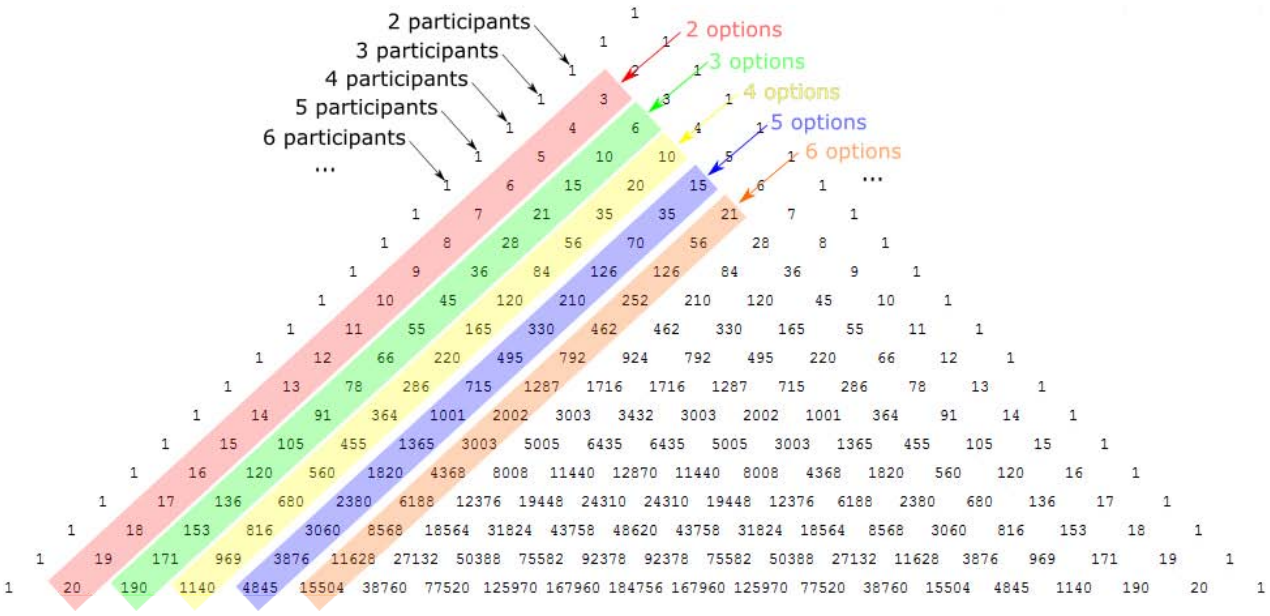

Figure 9.19.: Number of valid results in Pascal's triangle

One can see, that for the setup of this example (4 options, 6 participants), the number of valid permutations is in fact 84. Knowing that the value at the n^{th} line and k^{th} column in the Pascal's triangle (starting the count with 0) is given by $\binom{n}{k}$, it was found out that the complexity of this algorithm depending on o (number of options) and p (number of participants) is:

$$\binom{o+p-1}{o-1}$$

[7]Source: http://www.math.rutgers.edu/ erowland/images/pascalstriangle-large.gif

This result was already mentioned in the publication by Hao et al [6] describing the protocol implemented here.

In order to speed up the tally phase, all these possible results and their corresponding value that should be obtained by the protocol are computed in a background thread during the whole protocol run. As soon as the protocol initializations are done, this computation is started and the possible results and their corresponding value are stored in a hash map. Once the tally phase is started and the result of the protocol run computed, the program checks the hash map if the found result is already in the map. If so, the tally is terminated and the result can be displayed on the screen. If not, the program waits until the result is computed. The lookup is done via the Future concept of Java.

Multi encryption ballot This optimization although do not solve the increasing complexity for increasing number of options and participants. With some crypto protocols, there is a possibility to use multi encryption ballot instead of the Baudron et al encoding method. The idea in multi encryption ballot is, that a yes-no vote is generated for each option. So, a cipher text is generated containing a 1 in the exponent if the option was chosen. In the case of the present protocol, the encryption is not really an encryption but more a commitment. However, in this paragraph it will be considered as encryption since it the purpose of multi encryption ballot. So, cipher texts are created as follows: $b_{i_o} = h_i^{x_i} \cdot g^1$ where b_{i_o} is the encrypted vote of option o for participant i. If the option was not chosen, then a 0 is encrypted ($b_{i_o} = h_i^{x_i} \cdot g^0$) (see chapter 3 if a refresh of how the protocol works is needed). Thus, if there are o options, o cipher texts are computed for each participant. For each cipher text, a proof of valid vote must be computed to prove that the cipher text contains either a 1 or a 0. Moreover, a proof must be generated to show that a participant only chose one option. This can be done by computing a proof of validity showing that the sum of all encryptions produced by a participant is equal to 1.

The tally is then done independently for each option. The cipher texts corresponding to the first option are collected from all participants and are multiplied together. This results in homomorphically summing up the votes for this option in the exponent (obtaining g^{c_1} where c_1 is the number of votes for option 1). The same procedure is repeated for options 2 to o. So, at the end, a discrete logarithm has to be computed for each option, but these discrete logarithms are much more simple than the complex one obtained with the Baudron et al method. So, the multi encryption ballot method would help to make the tally of bigger votes faster. For small votes however, the Baudron et al method would stay the fastest.

However, the multi encryption ballot method seems not to work for the crypto protocol used in the present project. Recall that the encryption is done as follows: $b_i = h_i^{x_i} \cdot g^{v_i}$. When using multi encryption ballot, this operation should be repeated o times, once for each option. In $o-1$ cases, v_i would be equal to 0 and once v_i would be equal to 1. Since x, g and h_i are constant, this operation would give $o-1$ times the same cipher texts and once another. So, the vote secrecy could be broken only by watching the cipher texts, since the only one that is different of the others is the encryption of 1, thus corresponding to the chosen option.
In order to avoid this problem, a different value of x should be chosen for each option. However, this would require to compute o times a proof of knowledge for x_i, o times a h_i value, $o+1$ times a proof of validity as already mentioned, and, in case of the recovery round is needed, o times a $\hat{h}_i^{x_i}$ value and o times a proof of equality between discrete logarithms (see chapter 3 for the protocol description). As one sees, there are much more computations to do. So, the gain offered by multi

encryption ballot in the tally phase for this protocol would be compensated by the supplementary computations to do during the protocol run. This method would only be interesting for big votes where the tally process efficiency is very bad with Baurdon et al method. However, most of the boardrooms do not have the interest to run such big votes.

Method with precomputation Another possibility to optimize the tally process would be to do some precomputations for the permutations that must be tried out. With the standard method presented before, one modular exponentiation must be done for each permutation representing a valid vote. A way to improve that would be to consider each option separately. Each option can received maximum p votes where p is the number of voters. o represents the number of options. So, for a vote with 3 options and 4 participants for example, a vote would be encoded as follows:

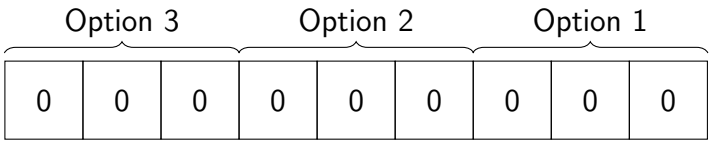

Figure 9.20.: Vote encoding using Baudron et al method

For each option, the cases can be considered when it receives 0, 1, 2, 3, or 4 votes and the corresponding values can be precomputed as show in following table:

	0 votes	**1 vote**	**2 votes**	**3 votes**	**4 votes**
Option 1	g^0	g^1	g^2	g^3	g^4
Option 2	g^0	g^8	g^{16}	g^{24}	g^{32}
Option 3	g^0	g^{64}	g^{128}	g^{192}	g^{256}

Table 9.1.: Example of precomputations for $p = 4$ and $o = 3$

One sees that when moving to the next column while staying in the same line, the value obtained for the previous column must be multiplied with the value obtained for 1 vote. For example, when computing the value for 3 votes, value obtained for 2 votes is multiplied with the one obtained for 1 vote. When wanting to compute the value for 1 vote for the next option, the value obtained for the previous line in the k^{th} column can be squared, where k correspond to the greatest power of 2 smaller than p. Values from column 0 votes are equal to 1.
Once this table computed, all permutations representing a valid vote must be tried out as done in the first method presented here. But this time, no modular exponentiation are needed anymore. They can be replaced with multiplications. For example, the result that should be obtained by the protocol if permutation is [2,1,1] can be compute by picking up, in table 9.1, the value corresponding to 2 votes for option 1 and multiplying it with the value corresponding to 1 vote for option 2 and with the value corresponding to 1 vote for option 3, namely $g^2 \cdot g^8 \cdot g^{64}$. This value can then be compared with the result obtained by the protocol.
With this precomputation method, one sees that instead of computing one mod exp for each permutation, only $o-1$ multiplications are required. However, the table 9.1 must be precomputed, requiring $o \cdot p$ multiplications. So, it seems to be very efficient, since no mod exp are required anymore. This can be interesting even for small votes. This method was also implemented in order to compare it with the previous mentioned.

9.4.7. Using Unicrypt

As already mentioned, for the implementation of the cryptography, the crypto library UniCrypt was used. This library was still under development. Some problems occurred when using it on Android. First of all, UniCrypt was written in Java version 7. However, Android only supports Java 6. Thus, the library had to be modified and the Java 7 constructs were removed.

Another point was the serializability of the classes. At the beginning of the project, UniCrypt was neither Java serializable nor had it another way to serialize the classes used here. So, it was decided to add Java serialization as it was the fastest way to have a working serialization. At present, another serialization method is under development.

Another problem faced was that the verification of the proofs worked on some devices while failing on some others. By studying the problem in more detail, it was noticed that the Android version played a role. The reason was a change in the implementation of the SecureRandom class since Android 4.2[8]. Since this version, it is no more possible to use a SecureRandom object to deterministically generate values. However, this feature was used in UniCrypt and had to be replaced.

9.4.8. Other Functionalities

An important thing in e-voting is the verifiability property which allows to check later that a voting process went of as planned. The protocol implemented here has this property and this check is already done during the whole voting process. However, it would be interesting to be able to check it again later, for example if someone claims that something went wrong. In order to be able to do that, an export function which creates an XML file containing all the cryptographic values generated in the protocol was implemented. This allows a third programmer to implement a program getting these values and replaying the protocol to check if the same result is obtained.

To export the values in XML, a library called SimpleXML[9] was used. This library has the disadvantage that it is quite big and considerably increases the size of the APK file[10] of the app. However, it presents the advantage to be able to directly serialize objects in XML. Since the values to write in the XML file are mostly UniCrypt objects and since there was no good way in UniCrypt to serialize objects, entities enabling to easily serialize and deserialize UniCrypt elements had to be created. This is especially true for proof elements which belong to complex mathematical groups. If there is a better serialization method in UniCrypt one day, this could be simplified. This modification, however, would make the verification process dependent on UniCrypt, what is not the case at present.

In order to show how this XML file could be used, a little verifier was developed. It is a shell Java program that reads the values of the XML files and verifies the proofs generated at each round of the protocol. Finally, it also checks if the previously computed result is correct. Another functionality is the ability to pass multiple XML files as parameters to the verifier program, which checks if the content of the files is identical. This allows to check if the protocol ran the same way on the device of different participants.

[8] http://android-developers.blogspot.ch/2013/02/security-enhancements-in-jelly-bean.html
[9] Simple XML serialization: http://simple.sourceforge.net/
[10] APK is the extension of the executable files on Android

This verifier uses the same code base as the implementation of the protocol in the app itself. So, this tool cannot really be used as official verifier but more as example of how the XML exported values can be used.

10. Results

This chapter reports the results of this work. Some general findings were already mentioned in chapter 5 in part 1 and, thus, are not repeated here. This chapter studies results that are more specific to the implementation.

10.1. User Interfaces

For what concerns the graphical user interfaces, most of the findings were already mentioned in part one. The most important however is probably that the same GUI could be used for the implementation of two different protocols. This was probably possible since the UI were studied to be used even without security. So, the implementation of the protocol could be done in a transparent manner.

The results of the usability tests showed that it is very important to ask people that do not know the project to try out the application. The feedbacks received were really constructive and some completely unexpected points and propositions came out. Most of the time, the analysis done by someone knowing how the application should work are not the most pertinent.

10.2. Load Tests

Some load tests were done in order to see how the network layer reacts and how the protocol scales. The first finding concerns the network layer. When setting up the first vote, only 16 voters could connect to the network group created by the administrator. After some research, it turned out that AllJoyn limits the number of client connections to 16 in Android. This is hard coded in AllJoyn's sources. Wanting to modify it requires to recompile the whole source, which is a complex process since the source of Android itself as well as the Android native development kit[1] must be used. It seems that there exists a workaround using a debug build of AllJoyn, where the configurations can be overwritten. However, not enough details about this workaround were available to use it in this case. So, the application is currently limited to 17 voters.

Another problem occurred with an Android programming concept. To pass data from an activity to another in Android, it is recommended to use an intent where the data to send can be attached as extras. The data are serialized and sent to the new activity. However, the size of these extras is limited. In the present case, at the end of the tally process, the whole protocol data must be sent to the activity displaying the results. The protocol data, however, contain multiple values of 2048 bits. The number of such values depends on the number of options and, above all, on the number

[1] Toolkit allowing to program in native languages like C and C++

of participants. Moreover, there is a complex structure of objects from UniCrypt that must be serialized in order to be sent. The fact was, that with all this stuff, the limit was exceeded. So, this problem was solved by giving a reference to the protocol object instead of sending it through the intent.

Another characteristic was tested, namely the number of clients that can connect on the built-in hotspot of an Android device. This clearly depends on the hardware of the device. On the Nexus 4 from Google, up to 10 clients could connect. On the Nexus 5, a try was made with 25 devices and they could connect without any problem. This means that if one of the voters owns a recent device, an external hotspot is no more required for a vote. So, the application can be used completely independently of external hardware.

The load tests also served, and this was their main goal, to evaluate how efficient the crypto protocol is on mobile devices. Each round of the protocol was considered. The conclusions that can be derived from these time measurements are described in following paragraph. As a reminder, values of a size of 2048 bits are used in the protocol.

- **Setup round**: no heavy task (in comparison with other tasks required by the protocol) is done in this round. The time needed to complete this round depends on the number of participants, because each of them has to send a message and process the ones received from the others. To be concrete, it took maximum 10 seconds on the least efficient device when 17 devices were participating.
- **Commitment round**: this round is not interesting since it requires a user interaction which lasts an arbitrary long time. The time for the commitment round depends on the time used by all users to vote.
- **Voting round**: the complexity of the voting round can be reduced to the complexity of the verification of the proof of valid vote. Table 10.1 shows the time needed on different devices to verify one proof of validity for a vote with different numbers of options. The time for this round then depends on the number of participants since there is one proof of validity for each voter.
- **Recovery round**: the recovery round is very similar to the setup round. The time needed for this one is even less than for the setup.

Device	5 options	10 options	20 options
Nexus 5	1.3 s	2.4 s	5.6 s
Nexus 4	2.0 s	3.9 s	8.7 s
HTC One V	2.8 s	5.8 s	24 s

Table 10.1.: Time required for verifying one proof of validity

As mentioned in chapter 5, the tally is the heaviest part of the protocol. Table 10.2 shows how much time different devices needed to compute one permutation. Using the formula to compute the number of possible permutations for a vote with o options and p participants,

$$\binom{o+p-1}{o-1}$$

the worst case time can be easily computed for all types of devices. The worst case in this situation means that the last computed permutation corresponds to the result obtained in the vote. The values shown here correspond to the standard method described previously.

Device	Time [ms]
Nexus 5	1.79
Samsung Galaxy Note 10.1	2.32
Nexus 4	2.54
Samsung Galaxy Tab 2 10.1	4.27
HTC One V	6.3

Table 10.2.: Time to compute one permutation in the tally process

So, one can conclude, that, with the standard method, if a limitation is fixed to 180 seconds (3 minutes) for the tally process, on the HTC One V device, a vote with 9 options and 9 participants could be realized. On a Nexus 5 device, 10 options and 10 participants would be supported. Of course, reducing one of the two parameters allows to increase the other, so with less options, more participants can be tolerated and vice-versa. However, the number of options has a greater impact on complexity than the number of participants.

Assuming that in most of the cases, up to five options are needed, up to 18 participants would be supported on the HTC One V device, and up to 24 on the Nexus 5. These are reasonable numbers that make this application usable in most of the cases. Keeping in mind that all these calculations were made for the worst case, these numbers could still be increased when assuming that an average of $2/3$ of the permutations must be computed to find the result obtained in the vote.

With the implementation of the method with precomputation also described in 9.4.6, a speed up could be reached. By simply computing the table of precomputations, the tally process takes between 2 and 3.5 times less time that with the standard method regardless of the number of options and participants the vote contains. The measurements could however not be realized on mobile device because of lack of time. However, it can be assumed that the speed up would be the same. So, with this method, the application is even more usable.

Following table shows the speed-up that could be reached for different number of options and participants:

	Participants			
Options	**5**	**10**	**15**	**20**
7	3	2.33	3.15	3.3
10	2.65	3.5	-	-
14	2.28	1.9	-	-
18	3.8	-	-	-

Table 10.3.: Speed-up with precomputations

This table gives an idea of the improvement made with the precomputations method. The value in the table are the ratio between the time needed by the standard method and the time needed by the precomputations method. As the table shows, there are variations, but no real tendency can be identified. Some values could not be computed because of lack of time. These time measurements should be done again in a more systematic manner. The ones here were only done to give an idea of the improvement obtained. There was no more enough time to make a better analysis.

10.3. Comparison with the Concurrent Thesis

The result obtained by Jürg Ritter in his thesis is different in term of efficiency of the crypto protocol. With the CGS97 protocol [2], both single encryption ballots method (with Baudron et al encoding) as well as multi encryption ballots method can be used. The second requires a little more computation and is not very good for small votes but rapidly becomes more interesting when the number of options and participants increases. The trade off seems to be about 10 options and 10 participants.

When using multi encryption ballot method in the case of the present thesis, the trade off would be reached later because of the amount of supplementary computations required to keep the vote secrecy property. Thus, it is less interesting in this case. However, the protocol implemented in the present case is more lightweight in term of computations done during the voting period.

For what concerns the user interfaces, the result obtained in these two theses is comparable since the same GUI was used. Only a little adaptation had to be made in the GUI of the CGS97 implementation since a threshold must be parameterized to ensure the robustness of the protocol.

11. Future Work

The application obtained in this project allows to set up a vote with up to 17 participants. There are however some functionalities that could be improved or that are not available yet, but would be useful.

Network layer The networking layer is especially concerned. Presently, the serialization used in the messages exchanged over the network is Java serialization. This is not the best solution if it is foreseen to adapt the application for non-Java platform. Two other improvements concern AllJoyn. For an unknown reason, AllJoyn limits the number of clients on Android to 16, this means that only 17 persons (including the administrator of the vote) can participate to a vote. This limitation is hard coded in AllJoyn's sources. Increasing this number would be a first improvement. The other one concerns the problem of finding the AllJoyn network group on some devices. Sometimes, some devices cannot connect to the network group because they are not able to find it on some networks. On other networks, it works fine. The reason for that is still unknown.
Another useful feature still concerning the network layer is the ability to recover the missed messages if a device temporarily disconnects from the network. Currently, this is not supported, so if the network connection of a device is interrupted, this device is no more able to participate to the vote. Implementing a recovery functionality would increase the robustness of the app.

Protocol efficiency The next optimization concerns the protocol. Currently, the tally of a vote with a big number of options and with many participants is slow. A solution should be found in order to better support such use-cases. A variant would be to use distributed computations. Each of the participants computes some of the combinations and when one finds the right combination, they send it to all the other participants, who can verify if it is the correct one. Another solution would be to merge various protocols in the same app, for example a more efficient one for small votes and a more efficient one for big votes and to dynamically choose the appropriate one. For example, the app obtained in the concurrent thesis could be merged with the one obtained here. Since the same GUI is used, this could be done quite easily. This could be interesting since the crypto protocol of the concurrent thesis is more efficient for bigger votes.

Currently, with the implemented method, there is also a risk to fill the whole memory of the device in the tally round. The actual implementation puts each valid permutation in a hash map. If the number of these results is big enough and if the voting process lasts enough time, a overflow of the available memory could happen.

Verifier The actual implementation of the verifier is very basic and uses the same code basis and the implementation of the crypto protocol itself. The implementation of an independent and more powerful verifier would be a significant advantage in the recognition of this app. By independent

is meant that another person or institution than the one that implemented the app should develop such a verifier.

Compatibility Last but not least, the actual application only works on Android devices. To be widely used in the mobile world, an app should also be available on at least on iOS devices like iPhones and iPads. This, however, would be a big job since the crypto library used is written in Java and iOS does not support Java. So, the protocol should be implemented without the help of this library. Moreover, the exchange of cryptographic values should be adapted.

12. Summary

The realization of this protocol is possible on a concrete mobile platform, however an adaptation of the threat model is needed. The user interfaces had to be adapted multiple times until a satisfactory result could be obtained after multiple tests done with project internal and external people.

As the limitation of the number of participants showed, using a third party software (AllJoyn in the present case) can be constraining, sometimes surprisingly since such limitations are not always documented.

When implementing a protocol, some security concepts, which are described with only a few words on the paper, are sometimes not so simple to translate in practice. A lot of different small aspects must be taken into account. This was for example the case for the authenticated network layer.

The implementation of the protocol confirmed the assumption of the complexity of the tally process. However, the product obtained seems to be usable for most of the common cases in boardroom voting.

13. Conclusion

This thesis showed that the protocol described in the publication titled *A Fair and Robust Voting System by Broadcast* can be used in practice on mobile devices. However, there is a limitation of the number of participants and the number of options the vote proposes. This is due to the heavy computations that must be done in the tally phase. Some improvements could already be done in order that the application is usable for most of the use-cases for boardroom voting.

Dedicating a part of this work for studying the user interfaces was a good idea. The results obtained on that level are good especially in sense of reusability of the same GUI for different protocols. As mentioned earlier, this part was done by two students together. This was a good idea since the work could be divided up. For the implementation of the protocols, having another student working on a similar project with a different crypto protocol was interesting for exchanges of ideas and discussions.

The realization of this project was for me the first confrontation with implementation of cryptography. This project also allowed me to deepen my knowledge in Android programming especially for the user interfaces design.

The possibility to use UniCrypt was also a benefit for this project, since the implementation of cryptography was facilitated. However, the fact that it is still under active development was a little bit annoying since updates had to be made regularly and they required changes to be done in the code of the application.

The general conclusion of this master's thesis is very positive. The project is not finished, it still has improvement potential, but it is in a state that it can be used on a wider scale.

References

[1] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical multi-candidate election system," in *PODC*, 2001, pp. 274–283.

[2] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," in *Advances in Cryptology — EUROCRYPT '97*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, vol. 1233, pp. 103–118. [Online]. Available: http://dx.doi.org/10.1007/3-540-69053-0_9

[3] S. Delaune and S. Kremer, "Formalising security properties in electronic voting protocols," Deliverable AVOTE 1.2, (ANR-07-SESU-002), Apr. 2010, 17 pages. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/avote-d12.pdf

[4] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986, pp. 186–194.

[5] R. Haenni and O. Spycher, "Secure internet voting on limited devices with anonymized dsa public keys," in *Proceedings of the 2011 conference on Electronic voting technology/workshop on trustworthy elections*, ser. EVT/WOTE'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 8–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028012.2028020

[6] F. Hao, P. Y. A. Ryan, and P. Zielinski, "Anonymous voting by two-round public discussion," *IET Information Security*, vol. 4, no. 2, pp. 62–67, 2010.

[7] D. Khader, B. Smyth, P. Y. A. Ryan, and F. Hao, "A fair and robust voting system by broadcast," in *Electronic Voting*, 2012, pp. 285–299.

[8] J. Ritter, "Instacircle – a location-bound ad-hoc network for android devices," Bern University of Applied Sciences, Engineering and Information Technology, Tech. Rep., 02 2013. [Online]. Available: http://e-voting.bfh.ch/students/theses-reports

[9] ——, "Decentralized e-voting on android devices using homomorphic tallying," Bern University of Applied Sciences, Engineering and Information Technology, Tech. Rep., 02 2014. [Online]. Available: http://e-voting.bfh.ch/students/theses-reports

A. User handbook

This chapter describes how the MobiVote Application can be used in practice. This part was written by Jürg Ritter. Since the use of the application is the same for both protocol implemented, this handbook was reused here after a little adaptation.

A.1. Purpose of this application

MobiVote can be used as an e-voting system for a small group of people. A possible use case would be a vote in a board of directors of a company, or any other committee where a group of people want to seek a decision in a secure manner. With MobiVote, such a vote can be set up very spontaneously as no special infrastructure is required. The system guarantees the privacy of each voter. It is not possible to find out how a participant has voted.

A.2. Prerequisites

In order to run a vote using MobiVote, all the participants need to be in possession of a mobile device such as a smartphone or a tablet computer running at least Android 4.0 (Ice Cream Sandwich). All the devices also need to be WLAN capable and the WLAN adapter has to be activated. Currently it is not possible to use mobile devices of other flavours such as Apple iPhones or devices running Windows mobile. In order to carry out a vote, at least two persons with their device must be involved.

A.3. Installation

The MobiVote application is available as an APK file which can be installed on Android devices. In order to install the APK file, the file needs to be transferred to the device. The easiest way to do so is to connect the device to a computer using the USB cable and copy the APK file to the device. Using a filemanager on the mobile device, locate the APK file in your filesystem and install it by clicking on it. For this to work, the option Settings » Security » Unknown Sources must be enabled.

A.4. Start of the application

After a successful installation, the app should be available in the applications screen of your Android device. The app can be started by clicking on the icon. This leads you to the main screen of the application (A.1).

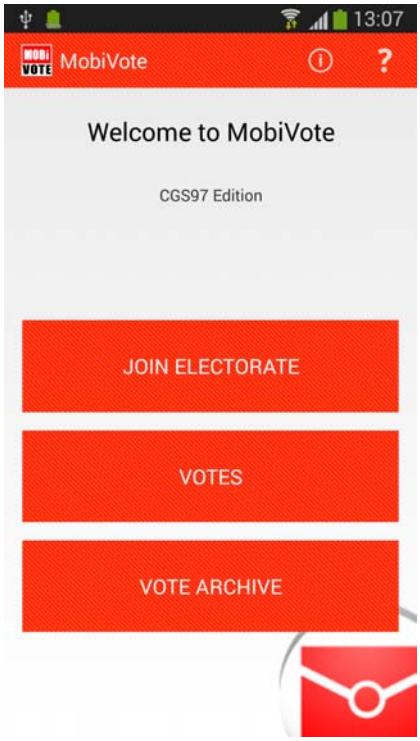

Figure A.1.: MobiVote main screen

A.5. Setup a vote

Aministrator only The group needs to nominate one member as the administrator of the vote. The administrator is responsible for the orchestration of the voting process. That includes defining the question and the possible options, approving other people to be in the electorate and initiating the voting phase.

Aministrator only In order to setup a new vote, the nominated administrator selects the option Votes on the main screen. The appearing screen shows a list of prepared votes, as well as an option to create a new vote (A.2). In the "Vote Setup" screen (A.3), the properties of the vote can be defined, namely the question of the vote and a list of possible answers. The option Allow blank ballots can be checked if voters should be allowed to cast a blank ballot.

From here you can either go back if you just want to prepare the vote in advance for an upcoming session, or you can directly start the vote by clicking on the button Start vote.

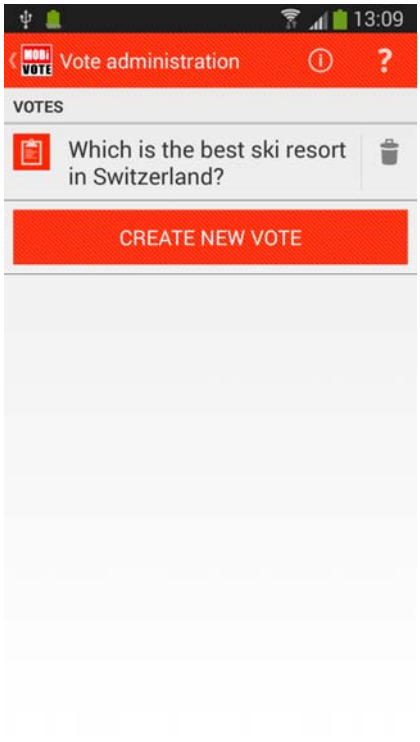

Figure A.2.: Available votes

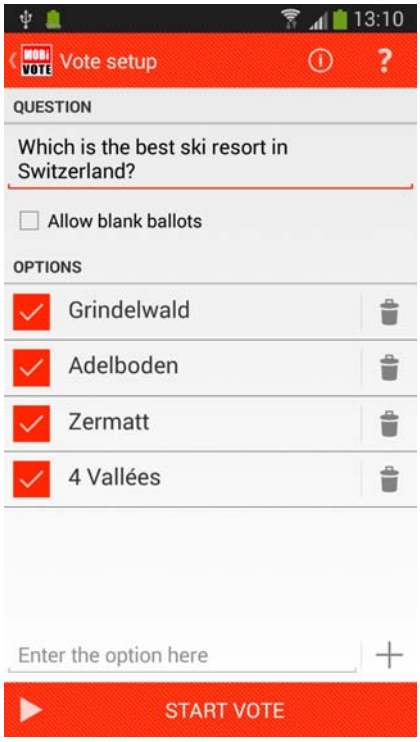

Figure A.3.: Vote setup

A.6. Start a voting session

Aministrator only In order to establish a voting session, some additional information regarding the network is required. First, you need to enter an identification into the according textbox at the top of the screen A.4. This value defines how you will be represented in the voting session. Most likely you will want to enter your name into this box. This value will be saved on the device and will be proposed in upcoming sessions, but you can change this representation anytime you want to. Moreover, you are required to tell the application in which wireless network the voting session should take place. In most cases this will be the network that you are currently connected to. If you want to use this network, you can just click on the button Use network "XYZ". In case you want to use a different network, you can use the button Advanced network configuration. This screen lets you choose all currently available networks (A.5). When choosing a network, you should consider that all the participants need to be allowed to connect to this network.

A.7. Share the session parameters

After selecting the network, a screen containing the voting session parameters is displayed (A.6). Three parameters are required to connect to a voting session:

- The network name (SSID)
- The group number
- The group password

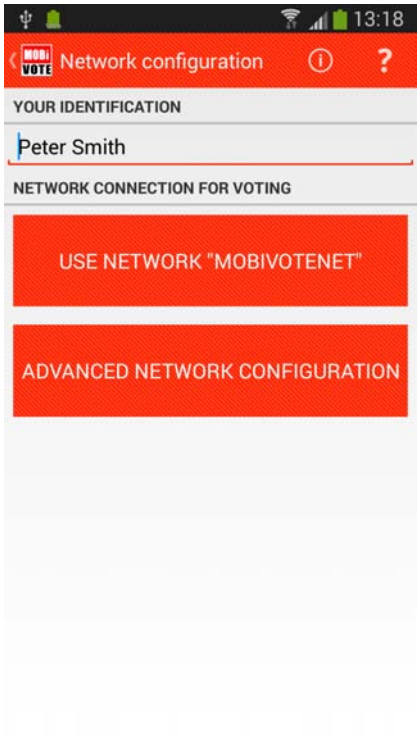

Figure A.4.: Network configuration

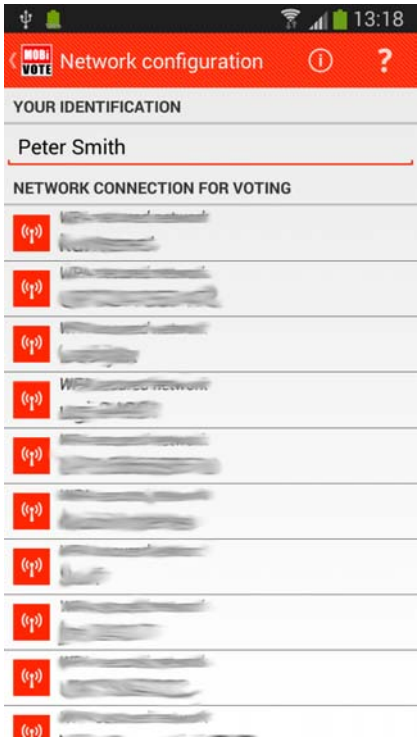

Figure A.5.: Advanced configuration

These three parameters need to be distributed to all the persons who will be allowed to join the electorate of the vote. To do so, MobiVote provides three ways.

- **Plain text:** All three parameters are communicated visually or orally as plain text. The participants can enter them when joining a voting session.
- **QR-code:** The three parameters are encoded in a QR-code which is displayed on the same screen. This QR-code can then be scanned using the camera of the participant's devices.
- **NFC tag:** The three parameters are written to a NFC tag. A NFC tag is a small magnetic storage token. An example of a NFC tag is depicted in A.7. This tag can then be passed along the participants who can scan the tag by tapping it to the back of their device. Please note that NFC functionality is only available in devices of the latest generation.

Of course it is also possible to use a combination of these three options. Using one of the sharing methods above, the participants can now join the voting session.

A.8. Join a voting session as a participant

Voter only Once the administrator has announced the session parameters, the participants can join the session using one of the sharing methods outlined in the previous section. After starting up the MobiVote app, the participants now chooses the option Join electorate. First, you need to enter an identification into the according textbox at the top of the screen A.8. This value defines how you will be represented in the voting session. Most likely you will want to enter your name into this box. This value will be saved on the device and will be proposed in upcoming sessions,

Figure A.6.: Network information

Figure A.7.: NFC tag

but you can change this representation anytime you want to. Next, one of the following methods to join can be chosen:

- **Scan QR-code:** Using this method, the QR-code displayed on another device can be captured using the camera of the device. After the successful capture, the device is automatically joined into the voting session. In case the session takes place on a secure WLAN which is not known on the device, the user will be prompted to enter the WLAN key. This method is probably the easiest and fastest way to join a voting session.
- **Use currently connected network:** When choosing this option, the WLAN to which the device is currently connected will be used. The user will be prompted to enter the password and the group number (A.9). These values are both displayed on the device of the administrator who established the voting session.
- **Advanced network configuration:** This option provides you with a list of currently available WLAN configuration (A.10). In order to join the voting session, you need to choose the WLAN which is displayed on the screen of the administrator's device. You will then be prompted to enter the password and the group number according to the parameters provided by the administrator.
- **Scan NFC tag:** This option can be used if you want to use a NFC tag which has been passed to you from the vote administrator (A.11). After tapping the NFC tag to the back of the device, the device automatically connects to the voting session. In case the session takes place on a secure WLAN which is not known on the device, the user will be prompted to enter the WLAN key. Please note that this option is only available on devices on which NFC functionality is available.

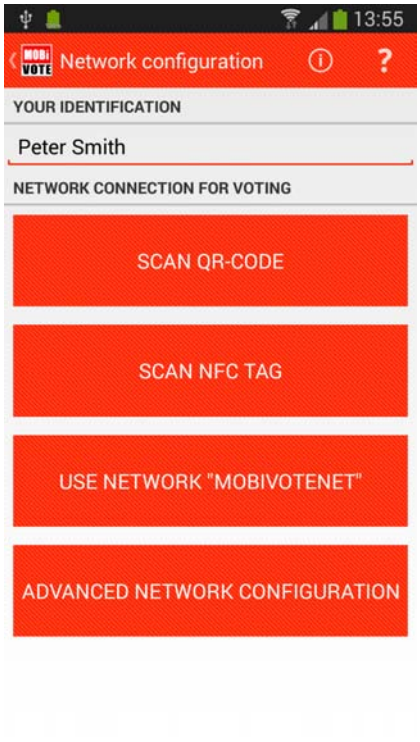

Figure A.8.: Join electorate

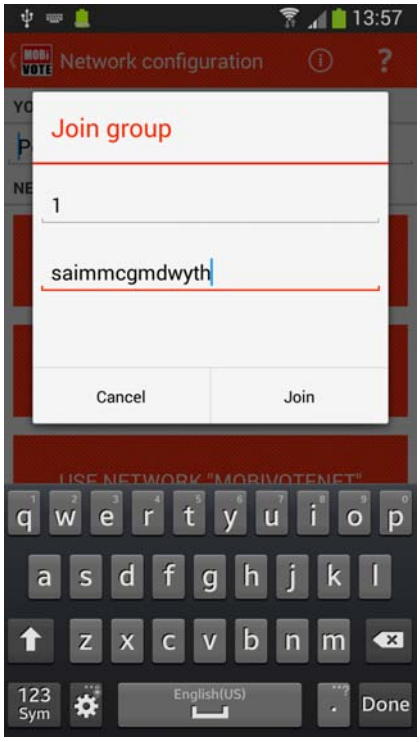

Figure A.9.: Enter password

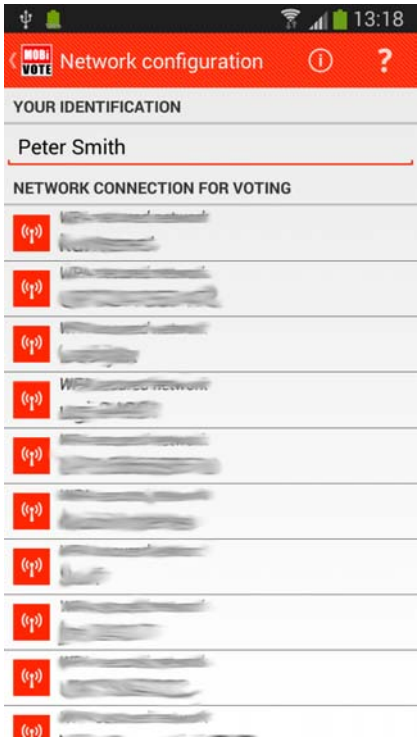

Figure A.10.: Advanced configuration

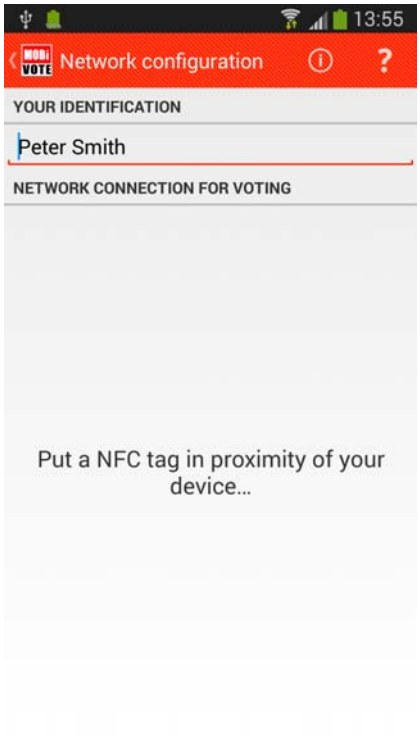

Figure A.11.: Scan NFC tag

A.9. Define the electorate

As soon as all participants were able to join their devices to the voting session, the administrator can forward to the next screen by clicking on Next. In the following screen A.12, the administrator can approve all the joined participants by clicking on the checkbox next to the allowed participants. Each action reflects immediately on all the joined devices (A.13). Please note that the administrator can include or exclude themselves in the electorate.

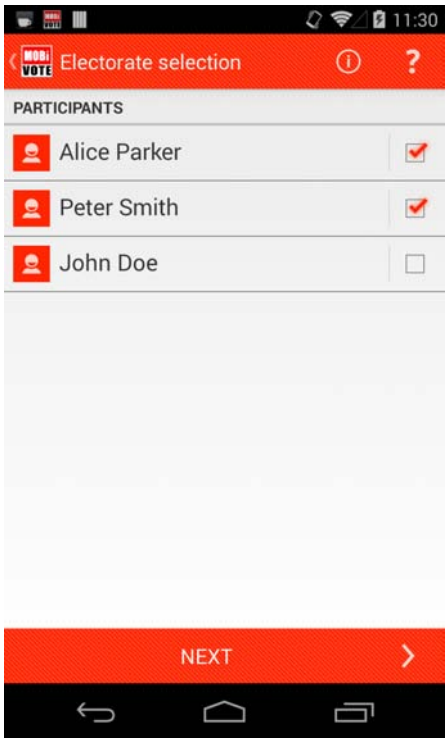

Figure A.12.: Admin defines electorate

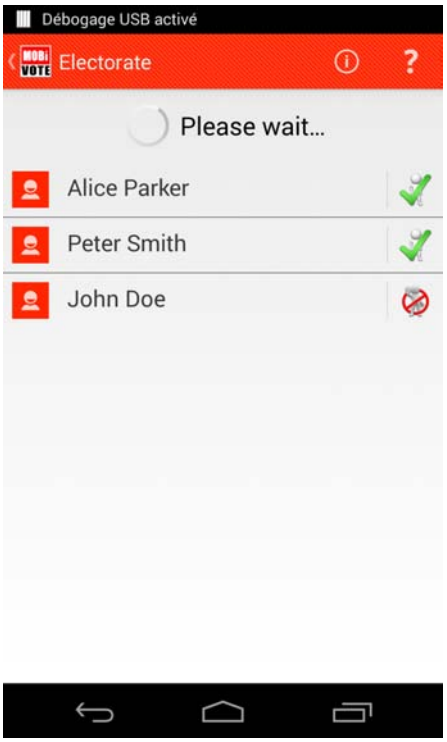

Figure A.13.: Display electorate

A.10. Review the vote

After having defined the electorate, the administrator can move along the process by clicking on the Next button. This leads to the review screen (A.14), where all the participants see a summary of the vote and its electorate. All participants have to agree on this setup by clicking on the checkbox next to their identification. In case some participants disagree, the administrator can always go back and adjust the electorate. Once all the participants have agreed on the setup, the administrator can start the actual voting period by clicking on the button Start voting period.

A.11. Voting

In the voting screen (A.15), each participant can select the preferred option. After confirming the choice, the vote is encrypted and cast. The following screen A.16 summarizes which participants cast their votes and which did not. The administrator has the option to interrupt by clicking on

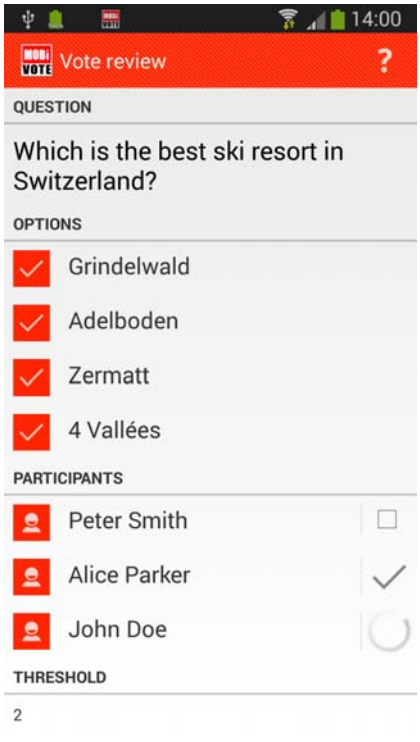

Figure A.14.: Vote review

Cancel vote, in which case no results are obtained. The administrator can also choose the option Finish vote, in which case the vote is tallied and the results are displayed. The tally of the vote starts automatically as soon as all participant have cast their votes.

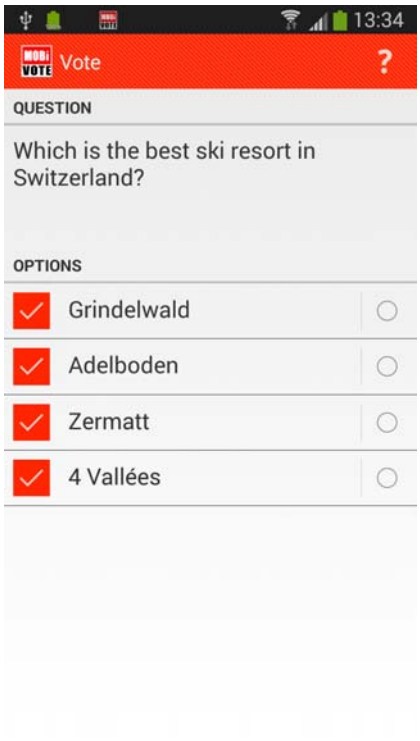

Figure A.15.: Vote screen

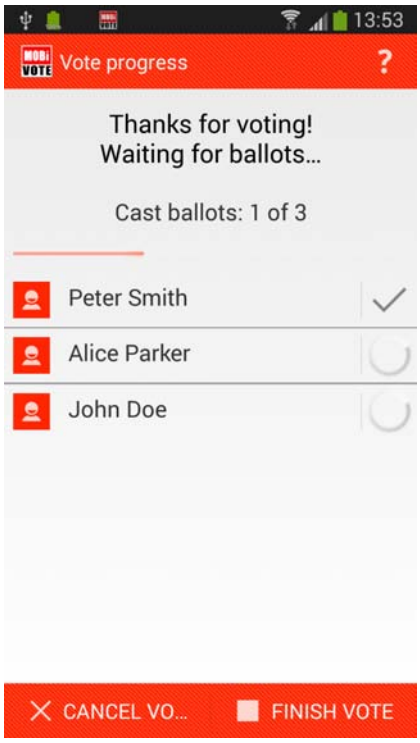

Figure A.16.: Waiting for ballots

A.12. Display the result

After a successful tally, you will be redirected to the result screen (A.17) where the result of the vote is displayed along with some statistics. The vote is archived and can be accessed anytime using the Vote archive option on the main screen. The administrator has the possibility to repeat the vote. This comes in handy if several voting round are required to reach a decision.

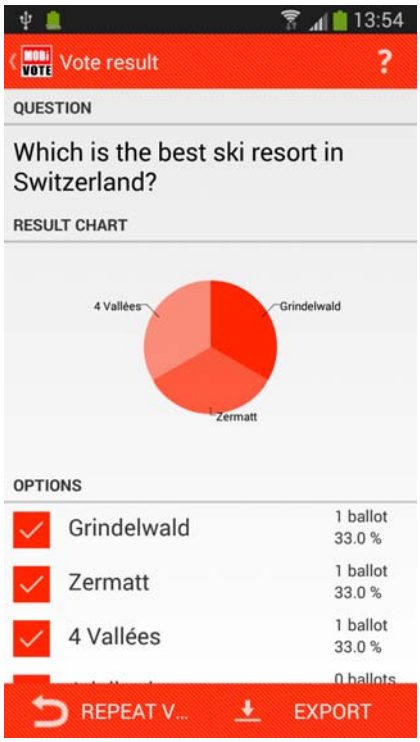

Figure A.17.: Result of the vote

A.13. Vote archives

The vote archive (A.18), which is accessible through the main screen, lists all the past votes. Unwanted votes can be deleted using the trash icon on the right side. An individual vote including the results can be examined in further detail by clicking on the according list item. If you want to run the exact election again, the Clone vote button can be used to recreate an empty vote which can then be started in the usual manner. The Export button allows to export the transcript of this vote to an XML file. This file could be used to verify that the vote has been done correctly.

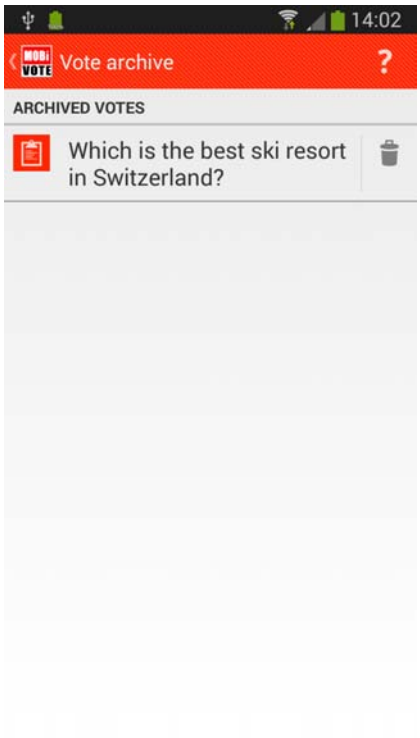

Figure A.18.: Vote archives