Master's Thesis

Decentralized E-Voting on Android Devices Using Homomorphic Tallying

Jürg Ritter

Bern University of Applied Sciences
Engineering and Information Technology
CH-2501 Biel, Switzerland

February 7, 2014

Advisor:
Prof. Dr. Rolf Haenni, Bern University of Applied Sciences

Expert:
Stephan Neumann, Technical University of Darmstadt

Abstract

During this master's thesis, a decentralized e-voting system for mobile devices such as smartphones and tablets running Android has been implemented. The term "decentralized" in that context means that there is no central server infrastructure involved in the voting process. The main application of such e-voting systems are votes with a low number of participants, for example the board of directors in a company. The idea is that the participants create an ad-hoc network with their mobile devices and then use this network to run a secure e-voting protocol. In order to guarantee privacy and verifiability, a voting scheme that uses homomorphic tallying is used. It also provides robustness in the sense that it is not possible for a single party to break the voting process just by refusing the collaboration at a certain step throughout the voting process. A big focus area is also the usability of this system, it should be possible for participants without extensive knowledge in the area of information security to use this e-voting system.

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:	Decentralized E-Voting on Android Devices Using Homomorphic Tallying
Firstname, Lastname:	Jürg Ritter
Date, Place:	February 7, 2014, Biel, Switzerland
Signature:	

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the E-Voting Group of the Bern University of Applied Sciences for giving me the opportunity to realize this project. Many interesting and inspiring discussions helped driving on this project. A special thanks goes to my advisor Prof. Dr. Rolf Haenni who supported me throughout the whole time during my master studies. A big thank you also goes to my fellow student Philémon von Bergen, with whom I worked closely during this master's thesis. Coming a long way from Germany for serving as an Expert for this master's thesis, I would like to express my gratitude to Stephan Neumann. I am also deeply grateful for the support of my employer Swisscom for supporting me during the whole time of my studies, especially my team who always gave me the necessary flexibility.

1 Introduction

The history of voting on a particular question in order to determine the will of a group of people goes far back to the ancient Greeks, who laid the groundwork of today's democratic societies. Over the years, the purpose of holding elections remained the same, while the procedures on how voting is done has changed significantly. The most ancient way of expressing the will by a hand sign has been replaced by modern technologies. The 20th century has seen a lot of changes in this area. Paper based voting has been introduced, and with the industrialization, the first voting machines appeared on the surface. The purpose of these machines was to simplify the counting process. In today's internet society where a lot of tasks such as banking, shopping, mailing, etc. have been revolutionized by the internet, it is not surprising that there are a lot of efforts going on to revolutionize the process of voting as well.

One of the research fields of the Bern University of Applied Sciences is the area of electronic voting, hereinafter abbreviated as e-voting. E-voting has become a big field of research in the past couple years. Still, there is no generic approach which meets all the criteria such as privacy, transparency, etc. which we want in e-voting. These properties are discussed in Section 2.1 on page 5. The e-voting research group of the Bern University of Applied Sciences[1] tries to improve this situation with the following approaches:

- Contribute to the scientific community in the area of e-voting by publishing new approaches.
- Take existing approaches and evaluate them in terms of practicability. These approaches are usually available as scientific papers.

The evaluation of these approaches is usually done by implementing them into a prototype level application. This implementation is used to show that the approach works in practice and provides a certain usability. The E-Voting Group would like to gain some experience on how decentralized e-voting systems could be implemented and how they behave in practice. A decentralized e-voting system aims to provide a platform to do secure e-voting without the need of having a central infrastructure (e.g. a server) available. There are some approaches which focus explicitly on this kind of e-voting systems, such

[1] `http://e-voting.bfh.ch/`

as the proposal of D. Khader et al. [10]. In this project however, we would like to adapt the voting scheme proposed by Cramer et. al. [3] in a way that it can be used as a decentralized e-voting system.

A potential use case of such a system could be boardroom voting, for example a vote held in a board of directors of a company. The architecture of our system requires that all the participants are in a confined space and are able to exchange some sort of credential using a communication channel which relies on physical proximity (e.g. near field communication) or visual contact (e.g. QR-codes).

In previous projects during the master studies, some groundwork has been implemented which can now be used as a foundation for this project. The previously implemented projects are the following:

- **UniCrypt:** UniCrypt is a cryptographic library developed by the members of the E-Voting Group of the Bern University of Applied Sciences. It provides cryptographic building blocks such as the ElGamal cryptosystem, zero knowledge proofs, digital signatures, etc.
- **InstaCircle:** InstaCircle provides a decentralized communication platform for Android devices. It allows to exchange messages using Wi-Fi communication among a closed user group.

The projects mentioned above will be discussed in more detail in Chapter 2 on page 5.

1.1 Goals

The final result of this project is a fully-fledged decentralized e-voting application which runs on Android devices and does not require any equipment other than the Android smartphones or tablets. The CGS97 e-voting scheme [3] will be used as the theoretical foundation for this project. Having said this, we divide the following goals for this project:

E-Voting Functionality. The main goal of this project is the implementation of the logic of CGS97 [3], by assembling the cryptographic building blocks to a working application. The final product needs to be able to handle a voting scenario *one-out-of-k*, meaning that a user can choose exactly one out of k options.

Graphical User Interface. In many cases, cryptographic functions which are heavily used in the context of e-voting have a negative impact on the usability. Many steps such as key generation, decryption, etc. have to be done in order to guarantee all the desired

properties for e-voting. All these steps bring a certain complexity into the handling of the application. Therefore it is crucial to guide the user through the process, and this can only be done by providing them a carefully crafted user interface. A further goal of this project is to implement a user interface which provides a good usability.

Extension of UniCrypt. UniCrypt implements many cryptographic building blocks which are necessary in order to implement a CGS97 based e-voting system, however two crucial parts are missing:

- **ElGamal Proof of Validity:** This type of zero-knowledge proof is used in order to make sure that a cipher text is an ElGamal encryption of an element which belongs to a set of possible plain texts. This type of proof is an instance of a more generic *OR-composition* of multiple proofs of knowledge. This type of zero-knowledge proof is discussed in detail in Section 2.2.5 on page 11.
- **Threshold Cryptosystem:** In order to ensure privacy of the ballots, the CGS97 e-voting scheme uses a so called threshold system. This system allows to split a private key and distribute *key shares* to a given set of participants. In order to decrypt a cipher text, the parties have to collaborate in order to reconstruct the private key. Threshold cryptosystems are discussed in further detail in Section 2.2.4 on page 9.

As a goal of this project, these two essential building blocks have to be implemented and integrated into the UniCrypt library.

The time budget of this master's thesis is one year, although the project will be implemented part time. It is equivalent to 27 ECTS credits.

1.2 Contribution

Currently there are a couple of Android apps available in the Google Play store[2] which allow to do simple votes, or else they are clients of e-voting systems of a specific organisation. A decentralized e-voting system which offers the cryptographic level of this project is currently not available. The implementation and the evaluation of this project will help to understand further strong and weak points of the CGS97 e-voting scheme, especially when using it in a decentralized configuration. So far, the E-Voting Group of the Bern University of Applied Sciences has gained expertise by implementing and operating e-voting systems

[2] `http://play.google.com/`

on centralized systems which are usually operated on notebooks or desktop computers. This project will help gaining some experience in how e-voting systems can be used on mobile devices.

Furthermore, the extensions for the UniCrypt library can be used for other projects as well. In the area of e-voting, threshold cryptosystems and validity proofs are considered being standard building blocks and can be used for the implementation of other e-voting schemes.

1.3 Outline

Chapter 2 on the following page gives an overview of the work on which this project has been based on. The cryptographic building blocks which we use are discussed, as well as the e-voting scheme on which this work is based on. We also discuss projects which have been implemented as a preparation of this master's thesis. The project setup and the administrative aspects are covered in Chapter 3 on page 26. Chapter 4 on page 30 explains the resulting implementation in detail. Chapter 5 on page 52 reflects on the work which has been done, before ending the document with a conclusion that summarizes the achievements (see Chapter 6 on page 58). A handbook of the product which has been implemented during this project can be found in the Appendix (see Appendix A on page 63).

2 Background and Related Work

This section gives an overview of the theoretical foundations and some projects which have been implemented earlier in order to serve as a foundation for this project.

2.1 E-Voting in 2014

Electronic voting has been a field of research ever since the internet revolutionized many areas of our daily life. Nevertheless, only few governments have made e-voting publicly available for their citizens. Some countries, among them Switzerland, are putting high efforts into officially introduce e-voting as an official way of voting, along with paper based voting. The reason why the introduction of e-voting takes much more time than for example e-banking stems from the fact that e-voting requires some properties which are not easy to implement. What follows is a non-exhaustive list of these properties.

Democracy: Only eligible voters can vote, and each voter can cast at most one ballot which will be included in the result.

Accuracy: The result is derived from *all* valid votes as they were cast, i.e. cast votes cannot be modified. No other factors than valid votes can influence the result.

Universal verifiability: Anybody, even people who are not involved in the voting process, can verify the correctness of the result.

Individual verifiability: A voter can verify that her vote is included in the result as she cast it.

Privacy: The e-voting system does not reveal any information about what option the voter has chosen.

Receipt-freeness: The voter cannot prove to anybody else how she voted.

Coercion-resistance: The voter cannot be coerced, meaning it is not possible to force a voter to vote in a particular way, or prevent a voter to vote at all.

Robustness: An attacker or a small colluding set of the electorate is not able to disrupt an election process or change the outcome by damaging a small set of system components.

All these properties are discussed in detail in [6] and [9]. The currently ongoing research efforts are put into developing approaches how e-voting systems that meet all these requirements could be built. Many of the properties mentioned above can be achieved by employing cryptographic building blocks, which are discussed in the next section. The robustness property is probably the biggest factor that prevents the breakthrough of e-voting on large scales. E-voting systems are the perfect targets for an intended manipulation or a boycott of an election. That is the reason why e-voting is currently used only on small scales. The size of the electorate is limited so that it is statistically unlikely to tip the scale to one side or the other.

2.2 Cryptographic Building Blocks

The voting scheme that is used in this project is assembled from some well known cryptographic building blocks which are briefly explained in this section. All these cryptographic building blocks belong to the area of *asymmetric* cryptography, which is usually based on a key pair, containing a *public key* and a *private key*.

2.2.1 Discrete Logarithm Problem

Asymmetric cryptography relies on a mathematical problem, which is nowadays considered to be hard to solve, meaning that so far there is no known algorithm to find a solution for a given problem efficiently. An example of such a problem is the prime factorization of large numbers on which the security of the famous RSA cryptosystem [16] is based on. The other hard problem of this kind is known as the *Discrete Logarithm Problem*. The formal definition of the problem is as follows:

Given a prime p, a generator g of a multiplicative modular group $\mathbb{Z}_p^*$ and an element $b \in \mathbb{Z}_p^*$, find the integer $x, 0 \leq x \leq p-2$, such that $g^x \equiv b \mod p$.

Currently there is no efficient way of solving this equation in an efficient manner. There are some more sophisticated approaches than the naive way of iterating over all elements of the group, for example the baby-step giant-step algorithm[1], but it is still not possible to find a solution in polynomial time with respect to the order of the group. The security of most of the building blocks which are used in this project is based on this problem.

[1] `http://en.wikipedia.org/wiki/Baby-step_giant-step`

Further information concerning the discrete logarithm problem can be found in Chapter 3 of the Handbook of Applied Cryptography by A. Menezes et. al. [12].

2.2.2 ElGamal Cryptosystem

The ElGamal cryptosystem [4], proposed by T. El Gamal in 1984, is the asymmetric cryptosystem which is mostly used in the context of e-voting. An asymmetric cryptosystem uses two keys to operate, one which is used to encrypt a certain message (the public key) and another to decrypt the message (the private key). The security of this cryptosystem is based on the fact that it is hard to compute the logarithm in discrete modular groups and hence making the exponentiation in modular groups a one way function. A major advantage of the ElGamal cryptosystem is the fact, that the encryption function includes a random value. Especially in the context of e-voting this is a crucial property, because when encrypting a value (for example the value 1 which means *yes*) multiple times using the same public key, the resulting cipher text is always different. That is why ElGamal is also called a *randomized* cryptosystem.

In order to define an ElGamal cryptosystem, three parameters are required. Let p and q be large prime numbers such that $q|p-1$.[2] The prime number q defines the order of a subgroup G_q of the multiplicative modular group $\mathbb{Z}_p^*$. $\mathbb{Z}_q$ is an additive modular group of order q. The last parameter needed is an arbitrarily chosen generator g of the group G_q. Further information concerning group theory can be found in Chapter 2 of the Handbook of Applied Cryptography by Alfred Menezes et. al. [12]. We can now derive the asymmetric key pair containing the private key $x \in_R \mathbb{Z}_q$ and the public key $y = g^x \in G_q$. A message $m \in G_q$ can be encrypted by first choosing a random value $r \in_R \mathbb{Z}_q$, and then applying the following function:

$$Enc_y : G_q \times \mathbb{Z}_q \to G_q \times G_q, (a, b) = Enc_y(m, r) = (g^r, y^r \cdot m)$$

The tuple (a, b) is the cipher text of the message m. m can be recovered using the private key x by applying the decryption function:

$$Dec_x : G_q \times G_q \to G_q, m = Dec_x(a, b) = a^{-x} \cdot b$$

Exponential ElGamal

In the area of e-voting, we often find a slight variation of ElGamal which is called *exponential ElGamal*. In exponential ElGamal, we encode a message $\hat{m}$ by using a

[2] In the case of $p = 2q+1$ where p and q are primes, p is called *safe prime* and q is called *Sophie Germain prime*.

generator $\hat{g}$ of G_q and raise it to the power of the message $\hat{m} \in \mathbb{Z}_q$. In many cases, the ElGamal parameter g is used as $\hat{g}$, which makes it unnecessary to define a fourth predefined parameter. Note that the message $\hat{m}$ is an element of the additive group $\mathbb{Z}_q$ as opposed to the classical ElGamal where m is an element of the multiplicative group G_q. The modified encryption function of exponential ElGamal looks as follows:

$$\hat{Enc}_y : \mathbb{Z}_q \times \mathbb{Z}_q \to G_q \times G_q, (a, b) = \hat{Enc}_y(\hat{m}, r) = (g^r, y^r \cdot \hat{g}^{\hat{m}})$$

In order to decrypt a given cipher text, we apply the ordinary ElGamal decryption function and obtain the value $\hat{g}^{\hat{m}} \in G_q$. To reveal $\hat{m}$, we would have to compute the discrete logarithm, which is a hard problem in general. If we have knowledge about all the possible plain texts (e.g. we know that a ballot either contains 0 for *no* or 1 for *yes*), we can just try all possible plain texts.

Exponential ElGamal has another property which is important in the context of e-voting: It transforms the right part of the ElGamal encryption function from a multiplicative homomorphic function into an additive homomorphic function. Homomorphic encryption is discussed in detail in Section 2.2.3.

2.2.3 Homomorphic encryption

The ElGamal cryptosystem has a property which is important for building verifiable e-voting schemes, namely the property of *homomorphism*. Given two mathematical groups $(X, \oplus)$ and $(Y, \otimes)$, a mathematical function $f : X \to Y$ is $(\oplus, \otimes)$-homomorphic if the following condition holds:

$$f(m_1) \otimes f(m_2) = f(m_1 \oplus m_2)$$

The ElGamal function offers exactly that property for its encryption function $Enc_y : G_q \times \mathbb{Z}_q \to G_q \times G_q$ using the same public key y:

$$Enc_y(m_1, r_1) \cdot Enc_y(m_2, r_2) = Enc_y(m_1 \cdot m_2, r_1 + r_2)$$

This property allows to calculate the encrypted product of all messages out of the encrypted messages and decrypt only the result. Applied to e-voting, the result can be computed out of the encrypted ballots and only the final result needs to be decrypted. The ballots themselves can remain encrypted, which is important for maintaining privacy. The ElGamal cryptosystem is homomorphic respective to the multiplication operation in the message part, which is not very fortunate for counting votes. Votes should be summed up in order to get the final result. In order to reach this, we use the following mathematical property:

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

This is exactly the property that we obtain by using exponential ElGamal as discussed in Section 2.2.2 on page 7. The product of all cipher text values results in the encrypted sum of the cast ballots, encrypted using the exponential ElGamal encryption function $\hat{Enc}_y : \mathbb{Z}_q \times \mathbb{Z}_q \to G_q \times G_q$:

$$\hat{Enc}_y(m_1, r_1) \cdot \hat{Enc}_y(m_2, r_2) = \hat{Enc}_y(m_1 + m_2, r_1 + r_2)$$

After decrypting the product of all cipher texts, we end up with a value of the form $\hat{g}^{(m_1+m_2)}$. The value that we are interested in is $m_1 + m_2$. This value can be obtained by computing the discrete logarithm, but we have seen in Section 2.2.1 on page 6 that this is a hard problem. In the context of e-voting, we can solve this problem by iterating over all possible solutions and compare it with the decrypted value. The number of possible solutions cannot be bigger than the number of cast ballots, therefore iterating over these solutions is not a big problem.

2.2.4 Secret sharing

In e-voting scenarios it is crucial that not a single entity can manipulate the result or reveal single votes. This responsibility, or in our case the private key which is needed to obtain the final result, has to be spread across a set of so called trustees. In the CGS97 scheme, this property is achieved by using a secret sharing mechanism as proposed by A. Shamir in 1979 [18]. This scheme even allows to define a so called *threshold*, which defines the minimal amount of participating trustees in order to decrypt the result. Such a system is also known as a $(t - n)$-threshold scheme, where n defines the number of shares which are issued at the beginning and t defines the minimal number of shares needed to recover the secret. Equal to the ElGamal cryptosystem, we need two large primes p and p such that $q|p - 1$. The prime number q defines the order of a subgroup G_q of the multiplicative modular group $\mathbb{Z}_p^*$. $\mathbb{Z}_q$ is an additive modular group of order q. Having defined these prerequisites, a secret can be shared as follows:

1. A trusted dealer chooses a polynomial $f(z) \in \mathbb{Z}_q(z)$ with uniformly random chosen coefficients $f_0, \ldots, f_{t-1}$. If the secret $x \in \mathbb{Z}_q$ has been defined earlier, the coefficient of degree 0 has to be equal to x.

2. The trusted dealer calculates a share $x_i = f(i)$ for $i = 1, \ldots, n$.

3. The trusted dealer *secretly* communicates the share x_i to each individual trustee.

The secret itself, at this stage only known by the trusted dealer, is defined by $x = f(0)$. In order to reproduce the secret using the shares distributed among all trustees, we

can use an interpolation technique such as Lagrange interpolation, which allows us to reproduce the secret $f(0)$. Since a polynomial function of degree $t-1$ is defined by at least t points, the secret can only be reproduced if at least t trustees are collaborating and contribute their shares.

The approach of Shamir is quite simple, but has one major drawback though: It requires a trusted dealer which has the knowledge of the secret. It it would be nice to have a scheme where the group of trustees collaborate to create the shares in a way that nobody can derive the secret unless a sufficient amount of trustees collaborate. In 1991, T. P. Pedersen proposed a scheme [13] where this trusted dealer is no longer required. In order to jointly generate a secret, the trustees perform the following steps:

1. All trustees τ_i for $i = 1, \ldots, n$ choose a polynomial $f_i(z) \in \mathbb{Z}_q(z)$ for $i = 1, \ldots, n$ with uniformly random chosen coefficients $f_{i0}, \ldots, f_{i,t-1}$.

2. All trustees commit themselves to their chosen coefficients by publishing the values $F_{ij} = g^{f_{ij}}$ for $j = 0, \ldots, t-1$.

3. All trustees generate a secret $x_{ij} = f_i(j)$ for each trustee $\tau_j, j = 1, \ldots, n$ and send it through a secure channel to τ_j.

4. All trustees verify that the received shares are consistent with the previously published coefficient commitments by verifying that the following equation holds:

$$g^{x_{ji}} \stackrel{?}{=} \prod_{l=0}^{t-1} F_{jl}^{i^l}$$

   If this verification fails, τ_i broadcasts a message that there was a problem, publishes the malicious share x_{ji}, and stops the protocol.

5. All trustees with correct shares can now compute their share x_i by computing $x_i = \sum_{j=1}^{n} s_{ji}$.

6. The public key y can be computed by calculating $y = \sum_{j=1}^{n} F_{j0}$.

The public key y can now be used by anybody, for example to do an ElGamal encryption. In order to decrypt an ElGamal cipher text (a, b) created using the public key y, at least t trustees have to collaborate and execute the following steps:

1. Each collaborating trustee τ_i publishes a part decryption $w_i = a^{s_j}$

2. Along with the part decryption w_i, each trustee has to publish a proof that testifies the following relation:
$$\log_g g^{x_i} = \log_a w_i$$
This proves that the trustee τ_i indeed used its key share to do the part decryption. Further details about zero-knowledge proofs can be found in Section 2.2.5.

As soon as at least t part decryptions are publicly available, anyone can use the Lagrange interpolation algorithm to recover the plain text. Let Λ be a set of at least t available shares. Using this set, the Lagrange coefficients can be calculated as follows:

$$\lambda_{j,\Lambda} = \prod_{l \in \Lambda \setminus \{j\}} \frac{l}{l-j}$$

The plain text m of the ElGamal cipher text (a, b) can then be recovered as follows:

$$m = \frac{b}{\prod_{j \in \Lambda} w_j^{\lambda_{j,\Lambda}}}$$

Note that a set of at least t trustees could recover the secret x just by running an interpolation algorithm. If $f(z)$ is the function recovered by interpolation, the secret x can be found by calculating $x = f(0)$. Recovering the secret is not desirable in the CGS97 voting scheme though, because a reunited secret key could be used to violate the privacy of the voters.

2.2.5 Zero-knowledge proofs

Zero-knowledge proofs (ZKPs) in their sense are conversations between a *prover* and a *verifier*. They allow the prover to demonstrate, that she knows a secret without actually revealing the secret itself. The conversation is similar to a challenge-response protocol. The verifier asks the prover certain questions about the secret and the verifier answers them. This kind of conversations are also known as Σ-protocols. Of course, the prover could just guess the correct answer to the question and cheat, but if the verifier repeats the challenge process with a different input, chances are almost zero that the prover can guess all the correct answers if she is not in fact in possession of the secret. Zero-knowledge proofs offer therefore a *probabilistic* security. In the context of e-voting, these zero-knowledge proofs are used to make sure that none of the participants are cheating.

Non-interactive zero-knowledge proofs

Zero-knowledge proofs as described earlier are *interactive* conversations between a prover and a verifier. This also means that the prover proves only to the verifier that she has knowledge of the secret. Of course, any observer could observe the conversation, but there is no way to determine whether the verifier actually accepts the the proof or not. The verifier could of course testify that the prover has the knowledge of the secret, but that would require a trust relationship between the observer and the verifier. In an e-voting scenario, we need a proof which can be verified by anybody and does not require an interactive conversation between the verifier and the prover. Such proofs are called *non-interactive zero-knowledge proofs* or *NIZKPs*. The foundations of these NIZKPs were introduced by A. Fiat and A. Shamir in 1986 [5], later known as the *Fiat-Shamir heuristic*. Instead of the verifier challenging the prover, the prover challenges himself by using a *hash function* such as SHA-256. As input for this hash function, publicly known values are used. A verifier can use these publicly known values later to verify that the challenge was created correctly. The result of such a non-interactive zero-knowledge proof is similar to a digital signature. Once published, everybody can verify the integrity of the data over which the signature has been calculated. In a similar way, non-interactive zero-knowledge proofs can be verified, with the important difference that the secret of course remains secret. So far, we did not specify what a secret actually is. There are multiple types of secrets and therefore also multiple types of zero-knowledge proofs, but due to the work of U. Maurer [11], we can formulate a general recipe how such a NIZKP is assembled:

Let $(X, \oplus)$ and $(Y, \otimes)$ be two mathematical groups and $f : X \to Y$ be a one-way homomorphic function. The prover wants to prove that she knows the preimage $\alpha \in X$ of the publicly known value $\beta \in Y$, where $\beta = f(\alpha)$. In order to prove the knowledge of the value α, the prover performs the following steps:

1. Choose a uniformly random value $\omega \in_R X$
2. Compute $t = f(\omega)$
3. Compute $c = H(\beta||t)$, where H represents a *hash function* such as SHA-256
4. Compute $s = \omega \oplus c \cdot \alpha$
5. Publish the proof $\pi = (t, s)$

A verifier can now calculate $c = H(\beta||t)$ and check whether the following condition holds:

$$f(s) \stackrel{?}{=} t \otimes \beta^c$$

In the following paragraphs, the different types of proofs used in this project are briefly explained and adapted to the generalized scheme explained before.

Proving the Knowledge of Discrete Logarithm

This type of proof was first presented by C. P. Schnorr in 1991 [17]. We have seen that the ElGamal encryption function has the homomorphic property (see Section 2.2.3 on page 8). Therefore, it is possible to prove the knowledge of the plain text of a given cipher text by applying the scheme above. We define the ElGamal encryption function using the public key y and the predefined generator g of the group G_q as follows:

$$Enc_y : G_q \times \mathbb{Z}_q \to G_q \times G_q, (a, b) = Enc_y(m, r) = (g^r, y^r \cdot m)$$

Since g and y are publicly known values, the knowledge of the value r implies the knowledge of m, therefore the prover only needs to prove the knowledge of r, which can be done by substituting the following variables in the generic scheme above:

$$\begin{aligned} X &= \mathbb{Z}_q \\ Y &= G_q \\ \alpha &= r \\ \beta &= a \\ f(x) &= g^x \end{aligned}$$

This translates to the following steps:

1. Choose a uniformly random value $\omega \in_R \mathbb{Z}_q$
2. Compute $t = g^\omega$
3. Compute $c = H(a||t)$, where H represents a *hash function* such as SHA-256
4. Compute $s = \omega + c \cdot r$
5. Publish the proof $\pi = (t, s)$

A verifier can now verify the knowledge of r and therefore m by calculating $c = H(a||t)$ and check whether the following condition holds:

$$g^s \stackrel{?}{=} t \cdot a^c$$

Proving the Equality of Discrete Logarithms

This type of proof due to D. Chaum and T. P. Pedersen [2] proves the relation $log_{g_1} c_1 = log_{g_2} c_2$ for two values $c_1 = g_1^m$ and $c_2 = g_2^m$, where g_1 and g_2 are generators of the mathematical group G_q. In order to prove this relation, we can again substitute the variables in the scheme above:

$$\begin{aligned} X &= \mathbb{Z}_q \\ Y &= G_q \times G_q \\ \alpha &= m \\ \beta &= (c_1, c_2) \\ f(x) &= (g_1^x, g_2^x) \end{aligned}$$

This translates to the following steps:

1. Choose a uniformly random value $\omega \in_R \mathbb{Z}_q$
2. Compute $t = (g_1^\omega, g_2^\omega)$
3. Compute $c = H(c_1||c_2||t)$, where H represents a *hash function* such as SHA-256
4. Compute $s = \omega + c \cdot m$
5. Publish the proof $\pi = (t, s)$

A verifier can now verify the relation by calculating $c = H(c_1||c_2||t)$ and check whether the following condition holds:

$$(g_1^s, g_2^s) \stackrel{?}{=} t \cdot (c_1, c_2)^c$$

Proving Validity

Proofs of validity are used to prove that a certain image of a homomorphic one way function is in fact the image of a preimage, and the preimage is an element of a set of possible preimages. The proof however does not reveal which preimage it actually is. This type of proof is not a straight forward application of the generalized scheme of Maurer [11], it is rather an OR-composition of several zero-knowledge proofs of knowledge. The idea is to *simulate* accepting conversations for the preimages which do not correspond to the calculated image and combine those simulated proofs with the actual proof from the image we calculated.

Let $(X, \oplus)$ and $(Y, \otimes)$ be two mathematical groups, $f : X \to Y$ be a one-way homomorphic function and $A = \{\alpha_1, \ldots, \alpha_n\} \subseteq X$ be a set of n possible preimages. The prover wants to prove that the publicly known image $\beta = f(\alpha_i) \in Y$ belongs to a preimage α_i without revealing which preimage it actually is.

Value Precomputation. In this phase, we need to compute a specially crafted image for all the possible preimages without the preimage for which the proof is being created. This can be done by executing the following step:

1. Compute $\beta_j = \beta \cdot \alpha_j^{-1}$ for $j = (1, ..., n)$.

Proof Generation. Now the proof can be created using the values b_j created in the precomputation phase, the chosen preimage a_i and the index i.

1. Select challenges $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n) \in_R X^{n-1}$
2. Select responses $(s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n) \in_R X^{n-1}$
3. Compute commitments $t_j = f(s_j) \cdot \beta_j^{-c_j} \in Y$ for $j = (1, \ldots, i-1, i+1, \ldots, n)$
4. Select $\omega_i \in_R X$
5. Compute $t_i = f(\omega_i) \in Y$
6. Compute challenge $c \in X$ with hash function[3] H: $c = H(\beta_1 || \ldots || \beta_n || t_1 || \ldots || t_n)$
7. Compute $c_i = c \oplus (\sum_{j=1, j \neq i}^{n} c_j)^{-1} \in X$
8. Compute $s_i = \omega_i \oplus c_i \cdot \alpha \in X$
9. Publish the proof $\pi = (t_1, \ldots, t_n, c_1, \ldots, c_n, s_1, \ldots, s_n)$

Verification. The verification step needs the same precomputation for all the values β_j. The proof itself can be verified checking that the following conditions hold:

$$f(s_j) \stackrel{?}{=} t_j \cdot \beta_j^{c_j} \text{ for } j = (1...n)$$

$$\sum_{j=1,}^{n} c_j \stackrel{?}{=} H(\beta_1 || \ldots || \beta_n || t_1 || \ldots || t_n)$$

[3] Usually a hash function such as SHA-256 is used, which does not necessary produce an element of X. Some sort of mapping is required.

The formal description above depicts how a general OR-proof is assembled. The cryptographic building block we need for our implementation is a proof of validity, which proves that the plain text m of a certain ElGamal cipher text (a, b) is an element of a set of possible plain texts $M = \{m_1, \ldots, m_n\}$. This scenario can be seen as an instance of an OR-proof. We can show this relation by substituting the variables in the generalized schema above as follows:

$$\begin{aligned} X &= \mathbb{Z}_q \\ Y &= G_q \times G_q \\ \alpha &= r \\ \beta &= (a, b) \\ f(x) &= (g^x, y^x) \end{aligned}$$

Note that as a function $f(x)$, we use the *identity function* of ElGamal, which encrypts the identity element of the domain (in the case of the multiplicative cyclic group G_q, the identity element is 1).

2.2.6 Bulletin Board

The so called bulletin board is the public communication channel which is used to communicate between the participants of the vote. It is a transcript of all the communication steps between the participants and therefore contains encrypted ballots, zero knowledge proofs, etc. The bulletin board is also available for observers. Using the content of the bulletin board, everybody can verify that all the participants are following the protocol or that dishonest participants are excluded from the voting process. A voter can verify that his/her own ballot is counted properly and also reflects in the final result. In theory, it is not possible to delete anything from a bulletin board (append only). There are several approaches on how to create such an append-only bulletin board. The CGS97 protocol itself only assumes this append-only property. Append-only bulletin boards are discussed in more detail in [8]. Since the bulletin board is a good target for a denial-of-service attack, it is a good idea to replicate the content of the bulletin board to multiple systems.

2.3 The Voting Scheme CGS97

In 1997, R. Cramer, R. Gennaro and B. Schoenmakers proposed a scheme [3] which allows to do e-voting in a secure and verifiable manner. The participants of the protocol can be divided into four different roles:

- **Administrator:** The administrator is responsible for setting up the election by defining the question and the possible options, the electorate and the voting period. The administrator is also responsible for orchestrating the activities during a voting cycle.
- **Voter:** A voter is somebody who is eligible to participate on a vote.
- **Trustee:** A trustee is somebody who helps setting up the election by creating an asymmetric key pair in cooperation with other trustees. At the end of the voting phase, the trustees have to cooperate in order to reveal the result of the vote.
- **Observer:** An observer is somebody who wants to verify that all the voters and trustees of a voting cycle behave as they are supposed to.

In the case of a decentralized voting system, the voter, trustee and observer roles can be combined and all participants impersonate these roles. The administrator role needs to be assigned to one particular participant.

2.3.1 The Protocol

The CGS97 voting scheme has a well defined procedure on how a vote should be performed. This procedure is explained in this section. Formally we define the set of n trustees as $T = \{\tau_1, \ldots, \tau_n\}$ and the set of l voters as $V = \{v_1, \ldots, v_l\}$. Furthermore, the role of an administrator A needs to be assigned to a participant. The process of these role assignments strongly depends on the political structure of the organisation which runs the vote and is considered to be an administrative task and is hence not further discussed at this point. The procedure how a vote is performed looks as follows:

Initialization Phase. All the parties need to agree on the ElGamal parameters p, q, and g. These parameters can be seen as fixed parameters and do not need to be redefined for each vote. Further details regarding these parameters can be found in Section 2.2.2 on page 7. Furthermore, the administrator A defines the question and all the possible options of the upcoming vote.

Key Generation Phase. All the trustees execute a robust threshold key generation protocol as discussed in Section 2.2.4 on page 9. At this stage, a *threshold* $t \leq |T|$ must be defined. This parameter defines the minimal number of trustees which are required to collaborate in order to decrypt the final result in the tallying phase. The transcript of the key generation protocol is posted on the bulletin board (see Section 2.2.6 on the

preceding page). The outcome of this key generation protocol is an ElGamal public key y which needs to be communicated to all the voters.

Voting Phase. During a well defined time window, the voters can now create and cast a ballot. To do so, the voters encrypt their ballots using the public key y and post them to the bulletin board. Note that we need to use *exponential ElGamal* (see Section 2.2.2 on page 7) so that we can *add* the result in an upcoming stage. Furthermore, each voter needs to post a proof of validity (see Section 2.2.5 on page 14) on the bulletin board in order to prove that the encrypted ballot indeed contains a valid option of the election.

Tallying Phase. The tallying phase can start as soon as the voting period has come to an end. In order to tally a vote, the cipher texts of the valid ballots are multiplied and posted to the bulletin board. Since all the encrypted ballots are publicly available, this multiplication step can be verified easily by doing the multiplication individually and compare the result to the values on the bulletin board. The product of all the valid ballots now represents the *encrypted result* of the vote. This result has to be decrypted by the trustees. The trustees execute a threshold decryption protocol in order to reveal the result of the election. Note that we only need a subset $\Lambda \subseteq T$ with minimal order t in order to execute the protocol properly. The transcript of the protocol and the result are posted on the bulletin board. Furthermore, each participating trustee has to provide a proof that he performed his decryption step in a correct fashion.

2.3.2 Voting Properties

In Section 2.1 on page 5 we discussed some security requirements which are desirable for e-voting systems. The following paragraph assesses the CGS97 protocol against these requirements.

Democracy. The CGS97 protocol does not specify how the access to the virtual voting booth is controlled. The e-voting system which implements the CGS97 protocol needs a sufficiently secure authentication mechanism in order to fulfill the democracy requirement. The criteria that only one valid vote can be cast is also tied to the authentication mechanism.

Accuracy. A zero-knowledge proof is used to prove that all valid ballots are tallied correctly. Since the all the relevant information is publicly accessible on the bulletin board, this can be verified by anybody.

Universal verifiability. The bulletin board which is publicly available (for voters as well as for observers who do not participate at the vote) assures verifiability for anybody. The bulletin board can also be seen as a transcript of the conversation between the actors during a voting cycle.

Individual verifiability. The voter can identify her own vote on the bulletin board along with the other votes. By verifying the zero-knowledge proofs of the trustees which are created during the tallying and decryption of the result, the voter can be sure that her vote has been included in the final result.

Privacy. The ballot which has been encrypted and cast by the voter always remains encrypted unless a sufficient amount of trustees decide conspire and decrypt single votes and reveal information on how the participants voted.

Receipt-freeness. A voter using the CGS97 scheme is able to create a receipt which can be used to prove to another person which option has been chosen. The CGS97 scheme hence does not offer receipt-freeness.

Coercion-resistance. Coercion resistant protocols have mechanisms to allow the voter to lie about the vote when under coercion. The voter pretends to vote according to the wish of the coercer while voting in fact according to her intention. Unfortunately, CGS97 does not offer this possibility.

Robustness. The CGS97 protocol offers robustness by including a threshold secret sharing mechanism that assures a working system as long as a certain number of properly behaving trustees remain in the system. The other crucial part which requires robustness is the bulletin board. To achieve robustness, the bulletin board should be replicated so that multiple consistent instances of the bulletin board are available. The CGS97 scheme does not specify in detail how this should be achieved. A possible approach can be found in [8].

2.3.3 Efficiency

From a voter's perspective, the CGS97 protocol is surprisingly efficient. No matter how big the electorate of the voting scenario is, there is only one interaction with the e-voting system in order to cast a ballot. This makes the protocol very efficient from a usability perspective.

The CGS97 voting scheme is also very efficient in terms of computation steps. When we consider a simple *yes-no* vote, the number of computation steps are linear with regards to the size of the electorate, as well as the number of trustees. The same is true for the trustees who are responsible for the key generation and the tallying process.

When we extend the voting scenario from a simple *yes-no* scenario to a *one-out-of-k* scenario, additional complexity comes in. The complexity depends on the encoding of the ballot, which is discussed in more detail in Section 2.4.

2.4 Ballot Encoding

In many cases, a voting scenario contains more than simply two options. In order to deal with a scenario of the type *one-out-of-k*, we need to think about the structure of a ballot which has to be able to contain more than just two options. To achieve this, we evaluated two strategies, both with their own advantages and disadvantages. These two strategies are discussed in the following.

2.4.1 Multi Encryption Ballot

In this strategy we treat each option as an individual *yes-no* question. To do so, an encryption containing a 1 for a *yes*-vote or a 0 for a *no*-vote for each individual option using exponential ElGamal is created. In order to prove that the encryption indeed contains either 0 or 1, a proof of validity is required for each individual encryption. Furthermore, we need to make sure that each ballot contains exactly one *yes*-vote. This can be achieved by creating a validity proof, proving that the sum of all encryptions is equal to one. In the tallying phase, a ballot is only accepted if all validity proofs are valid. For each option a homomorphic sum is calculated, containing the corresponding encryptions of all valid ballots. We then end up with k ElGamal encryptions, containing the result of each individual option. The trustees then perform a joint decryption of these values. Because exponential ElGamal is used, the decrypted results appear in the form $\hat{g}^{r_1}, \hat{g}^{r_2}, \ldots, \hat{g}^{r_k}$. To obtain the actual results $r_1, r_2, \ldots, r_k$, we would have to calculate the discrete logarithm, but as we have seen in Section 2.2.1 on page 6, this is a hard problem. A better way to obtain the results is to exploit the fact that we know the relatively small set of the possible results. Each result has to contain a value between 0 and n, n being the size of the electorate. Each result can now be probed by generating values $\hat{g}^0, \hat{g}^1, \hat{g}^2, \ldots$ and compare them to the values $\hat{g}^{r_1}, \hat{g}^{r_2}, \ldots, \hat{g}^{r_k}$ until the corresponding values for each option have been found. In the worst case, this decoding operation takes $k \cdot n$ steps. Figure 2.1 on the following page visualizes the growth of the number of steps as the electorate and the number of options grow.

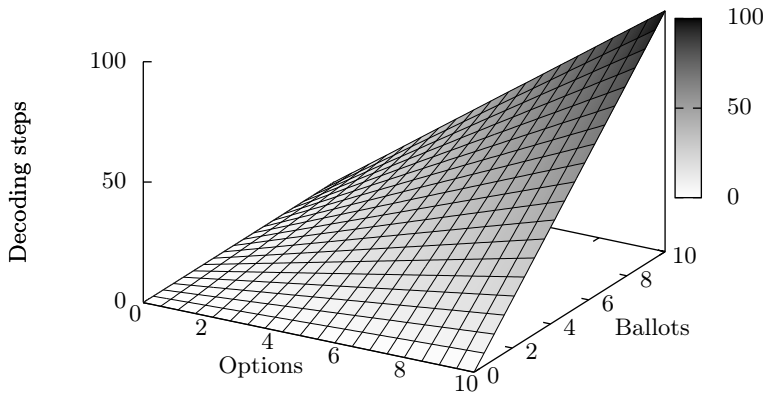

Figure 2.1: Multi encryption ballot - complexity

2.4.2 Single Encryption Ballot

In this strategy, the choice of each option is encoded in a single value, which is then encrypted. To do so, a bit zero vector which is divided into equally sized sections for each individual option is used. Let k be the number of options in a vote, and n be the size of the electorate. For this scenario, we need a bit vector with k sections, each containing at least $\lceil \log_2 n + 1 \rceil$ bits. A vote can then be encoded by flipping the least significant bit of the chosen option's corresponding section to 1. This creates a binary representation of the number $v \in \mathbb{Z}_q$, provided that the bit vector does not exceed the bit length of the order of the group. Figure 2.2 on the next page illustrates how such a vector would look like if someone chose option B. A voting scenario with this setup would be limited to $2^4 - 1 = 15$ participants because each sector can only accommodate numbers from 0 to 15. In order to achieve additive homomorphism with respect to the message, this number needs to be encoded by calculating $\hat{v} = \hat{g}^v$ (see Section 2.2.2 on page 7). This number can now be encrypted using ElGamal. In order to prove that exactly one option has been chosen, a proof of validity needs to be created along with the ElGamal encryption.

During the tallying phase, the encrypted values for each ballots are verified and the homomorphic product of all valid ballots is calculated. After the joint decryption of

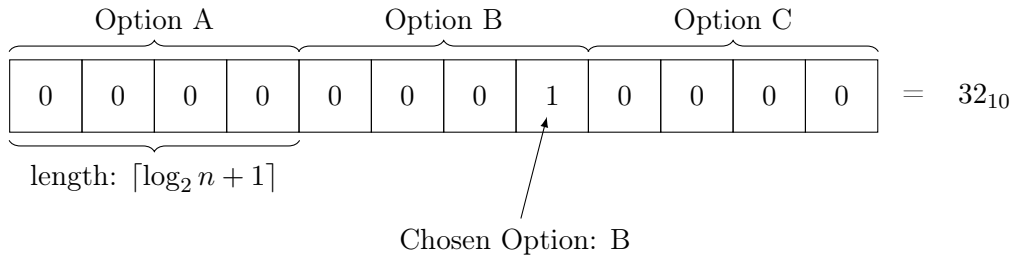

Figure 2.2: Single encryption encoding - ballot vector

this homomorphic product by the trustees, the result of the vote has the form $\hat{g}^v$, where $v \in \mathbb{Z}_q$ again represents a bit vector. Each sector of this bit vector now contains the sum of each individual option. This works because the dimensions of the sectors have been designed as such that they do not overflow while summing up the votes. Figure 2.3 illustrates the bit vector v containing each option's result after tallying the vote.

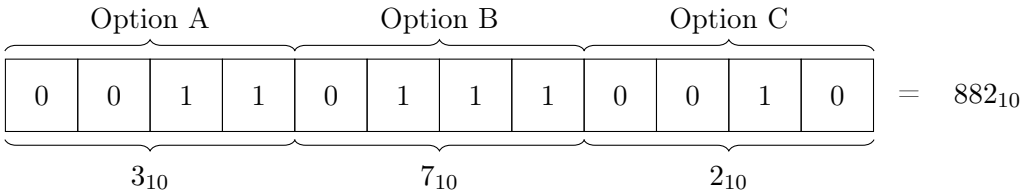

Figure 2.3: Single encryption encoding - result vector

We have seen that after decryption we end up with a value of the form $\hat{g}^v$. v could be obtained by calculating the discrete logarithm, but as we have seen in Section 2.2.1 on page 6, this is a hard problem. In this case though, we can exploit the fact that the number of votes is known. Using that fact, we systematically create probing vectors $c_1, c_2, c_3, \ldots$ where the sum of the values encoded in the sectors correspond to the number of votes that have been cast. We then check whether $\hat{g}^{c_i} \stackrel{?}{=} \hat{g}^v$ for $i = 1, \ldots, x$ until we found the corresponding vector. As we can see in Figure 2.4 on the next page, the number of steps to decode a vote is growing factorially as the number of participants and options go up. Using this method, decoding a vote with 10 participants and 10 options already needs almost 100 000 modular exponentiations.

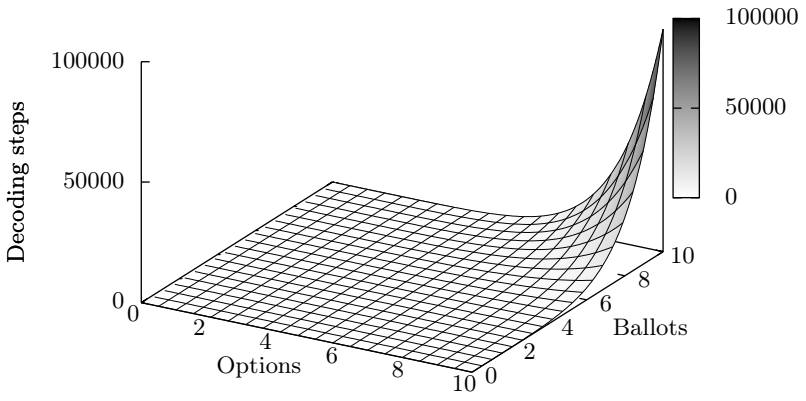

Figure 2.4: Single encryption ballot - complexity

2.4.3 Comparison

Each of the previously discussed approaches on how to encode a ballot has its own advantages and disadvantages. For comparing the two approaches, we are using a *one-out-of-k* voting scenario with k options and n participants. As we can see in Table 2.1 on the following page, the single ballot encryption is in many ways more efficient, except for the decoding part. As the number of participants and options grow, the number of combinations which have to be probed in order to decode the final result is growing quickly. For small voting scenarios ($n \leq 10, k \leq 10$), the single ballot encryption encoding is clearly the more efficient choice.

	Multi Encryption Ballot	**Single Encryption Ballot**
Encryptions / ballot	k	1
Validity proofs / ballot	$k+1$	1
Decryption operations	k	1
Number of combinations for decoding	$k \cdot n$	$\binom{k+n-1}{k-1}$

Table 2.1: Comparison of ballot encodings

2.5 UniCrypt

UniCrypt is the name of a cryptographic library which is developed and maintained by the E-Voting Group of the Bern University of Applied Sciences. The main goal of this project is to create a platform on which upcoming projects can be built on. UniCrypt tries to look at cryptographic functions such as cryptosystems, signature systems, hash functions etc. from a mathematical point of view. Cipher texts, plain texts, signatures, hashes, etc. are treated as elements of algebraic structures, the cryptographic functions are treated as mathematical functions. This is necessary because in secure protocols used in the area of e-voting, we depend on the plain textbook implementation of the cryptographic functions without functions like automatic padding, encoding, etc. This is the main difference between UniCrypt and other cryptographic libraries.

At the moment, UniCrypt has only been used in classic Java environments, but not on mobile devices. At this point, a re-engineered version of UniCrypt is under active development. Further information about UniCrypt can be found in [15].

2.6 InstaCircle

InstaCircle is the name of a project that has been implemented as a preparation of this master's thesis. It is intended as a platform to connect mobile devices in a confined space to a Wi-Fi ad-hoc network in order to exchange broadcast and unicast messages. In order to keep the network traffic as low as possible, the basic communication relies on broadcast messages. Since it is not possible to implement a reliable protocol using broadcast techniques, mechanisms to compensate message losses have been implemented. These re-sending mechanisms rely on a reliable unicast channel.

From a usability perspective, the process on agreeing to a communication channel for an ad-hoc network is quite challenging. All the participants need to have to know how to switch their devices to the correct Wi-Fi network and enter the correct keys and passwords. InstaCircle tries to improve the usability by allowing the users to exchange the configuration by sharing a QR-code or a NFC tag. Once a user has set up a conversation, other participants can join the conversation by scanning the QR-code or the NFC tag shared by the initiator of the conversation.

InstaCircle is currently available for Android devices only. Further information about InstaCircle can be found in [14].

2.7 Related Projects

During the same time period, Philémon von Bergen realized his master's thesis [1], implementing the same kind of application but with a different e-voting scheme. His application uses a protocol specifically designed for decentralized e-voting. It has been designed in 2012 by D. Khader et. al. [10]. A research team at the Technical University of Darmstadt is also planning to implement a similar application which is able to deal with several protocols and more complex voting scenarios.

3 Organisation

This project has been implemented as a master's thesis project during the Master of Science in Engineering studies (MSE) at the Bern University of Applied Sciences. This project values 27 ECTS credits and was spread among two semesters. The hand in of the project is scheduled for February 7, 2014.

During the same period, my fellow master student Philémon von Bergen worked on a very similar topic [1], namely the implementation of the decentralized e-voting system proposed by D. Khader et al. [10]. There was a big collaboration between the two projects, mainly for the implementation of the graphical user interface. The collaboration between these projects also made it easier to compare the approaches and evaluate the advantages and disadvantages of the two approaches.

3.1 Timeline

Original planning. The gantt chart in Figure 3.1 on the next page visualizes the steps which have been planned on the time axis during the project period. At first, we defined the project setup, later on the cryptographic components were put in place and were tested on the Android platform. It was followed by the storyboard and the implementation of the user interface. Especially in this part there was a strong collaboration with Philémon von Bergen [1]. At this point we had the cornerstone and were able to implement the actual e-voting related logic, the implementation of the CGS97 protocol. For this step we calculated the biggest time period.

Effective timeline. As in many project, there were small deviations regarding the time consumption of some implementation steps. The implementation of the cryptographic building blocks in the first part could be done as planned. The implementation of the graphical user interface and the integration of the network layer took more time than anticipated, we finished this part in week 46. In order to start with the implementation of the e-voting logic, we had a dependency on UniCrypt, which converged to a stable code base in week 46, so an earlier start of the implementation was not possible anyway. Because of the enormous help of UniCrypt, the implementation of the e-voting logic took much less time than anticipated, which helped catching up with the planning.

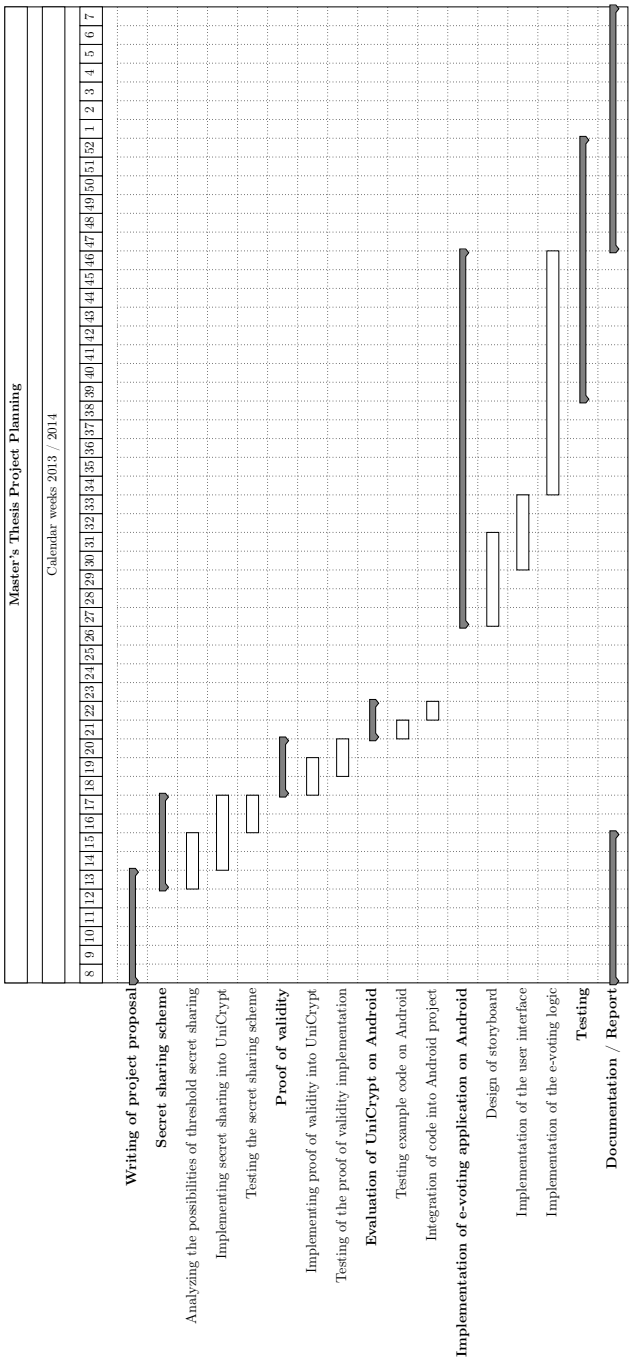

Figure 3.1: Project planning

3.2 Meetings

In order to synchronize the two projects regarding progress, problems and the coordination of the next work packages, we agreed on one day per week to work together at the same location in Biel. It was also used to catch up with the advisor and discuss the next steps.

3.3 Project Dependencies

It was planned to use the UniCrypt cryptographic library for implementing the e-voting logic. This has been done mainly for two reasons:

- Simplify the implementation by gluing together predefined cryptographic building blocks.
- Use the re-engineered version of UniCrypt for the first time in a project and provide feedback.

This creates a significant dependency to the UniCrypt project. The library has been under heavy development during this project and a lot of changes have been implemented even after we started with the e-voting logic. We could also provide some feedback which has been integrated into UniCrypt by the UniCrypt development team.

3.4 Source Code Organisation

As a source code repository and a collaboration platform, the free hosting platform GitHub[1] has been chosen. It provides Git[2] source code repositories which allow to do code versioning and branching. It also enables multiple contributors to maintain a certain code base, which was important during the implementation of the user interface and the network layer. Using the branching mechanism of Git, we were able to split the code base apart and implement the project specific protocols on top of the common code base. Table 3.1 on the following page outlines the three repositories which have been used for the implementation for this project.

[1] `https://github.com/`
[2] `http://git-scm.com/`

Component	Repository URL	Branch
Voter App	`https://github.com/jritter/VoterApp`	CGS97
AllJoynLib	`https://github.com/jritter/AllJoynLib`	unicastfeature
ZXing	`https://github.com/jritter/ZXing`	master

Table 3.1: Source code repositories

4 Results

During the project phase, a working implementation of the CGS97 voting protocol running on Android mobile devices has been implemented. We named the application "MobiVote", which is related to the name "UniVote", which is an e-voting system targeted for the use at a desktop computer. UniVote[1] is developed at the same institute, therefore we tried to create this relation.

In this chapter we present further insights of the design of MobiVote. We will discuss the technologies on which the implementation is based, the development of the graphical user interface, the architecture and implementation of the Android application and finally the contributions to related projects.

4.1 Used Technologies

This Section outlines the frameworks which have been used for the implementation of this project and how they are put together into an application.

4.1.1 Android

The Android[2] platform serves as the foundation of the whole implementation. The Application has been built using existing building blocks such as Activities, Services, Fragments, etc. Furthermore, there are existing guidelines and best practices which have been employed whenever possible. For developing Android applications, the popular Java[3] programming language can be used. The Android development kit then compiles this Java code and creates Android specific Dalvik VM byte code which is then executed on Android devices.

[1] `https://www.univote.ch/`
[2] `http://developer.android.com/`
[3] `http://www.java.com/`

4.1.2 AllJoyn

AllJoyn[4] is a library which allows peer-to-peer communication between several devices. The library has been released as open-source recently by the Qualcomm company. It is available for multiple platforms, including Android. AllJoyn serves as a communication layer which allows to exchange messages between devices such as smartphones and tablets. One device acts as master and starts advertising a group to which other devices can join. Once joined into the group, all the devices in the group can exchange messages among each other. The communication can happen either in a broadcast (one to many) or unicast (one to one) manner. The AllJoyn session is by default not secured at all, all devices in the same Wi-Fi network can join into a group. The security layer used for this project has been built on top of this insecure communication layer. The implementation is explained in Section 4.3 on page 38.

The original intention was to use InstaCircle [14] as a communication layer. It turned out that this type of completely decentralized communication layer did not suit our needs and did not work as expected in some network configurations. Compared to InstaCircle, AllJoyn bears the drawback that the communication is orchestrated on the master device that initiates the group communication. In exchange, the communication is reliable.

Some parts of InstaCircle were used, mainly the WLAN management and the logic for transmitting the communication parameters could be adapted from the InstaCircle implementation.

4.1.3 ZXing

ZXing[5] (pronounced "Zebra Crossing") is a Java library provided by Google which allows to deal with many flavors of bar codes, including QR-codes which are used in this project. It provides the logic to encode information into a QR-code and display it on the user interface of the application. It also provides the functionality which allows to use the camera of a smartphone/tablet as a scanner in order to decode the QR-code which is being displayed on the screen of another device.

In this project, we are using this functionality for transmitting the parameters of the group to other devices. These parameters include the SSID of the WLAN that is being used, the encryption key of the conversation, as well as a group identifier in case there are multiple conversations running in the same network.

[4] `https://allseenalliance.org/`
[5] `https://code.google.com/p/zxing/`

4.1.4 AChartEngine

AChartEngine[6] is an open-source library which can be used to draw all sorts of charts on Android devices. In this project, we use this library in order to visualize the result on the result screen using a pie-chart.

4.1.5 Simple

Simple[7] is an open-source framework which allows to serialize and deserialize Java objects to XML. It is used to save and export the transcript of a vote to a XML file. This functionality is further discussed in Section 4.5 on page 43.

4.1.6 UniCrypt

UniCrypt[8] is a cryptographic library which provides mathematical and cryptographic primitives which are used for implementing the e-voting protocol for this application. The version 2 of UniCrypt is currently under active development. The use of this library in this project also helped to improve UniCrypt itself. Since UniCrypt is developed at the same institute, problems could be addressed and fixed right away. Since this project is one of the first projects using UniCrypt, it also served as a test bed for UniCrypt. For this project, some non-existing components of UniCrypt have been implemented. The result of this work is discussed in further detail in Section 4.8 on page 50.

Since UniCrypt is an essential cornerstone of this project, it is discussed in further detail in Section 4.1.6.

4.2 Graphical User Interface

A crucial aspect for the success of e-voting is the usability for the end user. This becomes even more important when e-voting should be used in an ad-hoc manner, where the formality is less visible compared to official elections. Also, the time consumption for a voting cycle has to be as short as possible, which is also a significant difference to classically held elections.

[6] `http://www.achartengine.org/`
[7] `http://simple.sourceforge.net/`
[8] `https://github.com/bfh-evg/unicrypt/`

4.2.1 Storyboard

Before the actual implementation of the Android application, we tried to develop a screen design which meets these requirements. This screen design has been designed as a storyboard without writing actual code. We have chosen an iterative approach where the design has been peer reviewed several times before the final draft was available. One major decision point was to decide whether we implement a single Android application or whether we should split it into an app for the voter and another for the vote administrator. The approach with multiple apps turned out to be confusing and was hence dismissed.

The design which has been finally chosen is based on a single Android application. A voting cycle can be broken into several phases which are outlined in the following, along with the storyboard for this particular phase.

Create a Vote. The designated administrator of the vote is responsible for creating a vote, namely defining the question and the possible options which can be chosen later during the voting phase. A vote can also be prepared and saved in advance in order to save time at the reunion. From here, the administrator can also start the voting process. The screen design of this phase is illustrated in Figure 4.1.

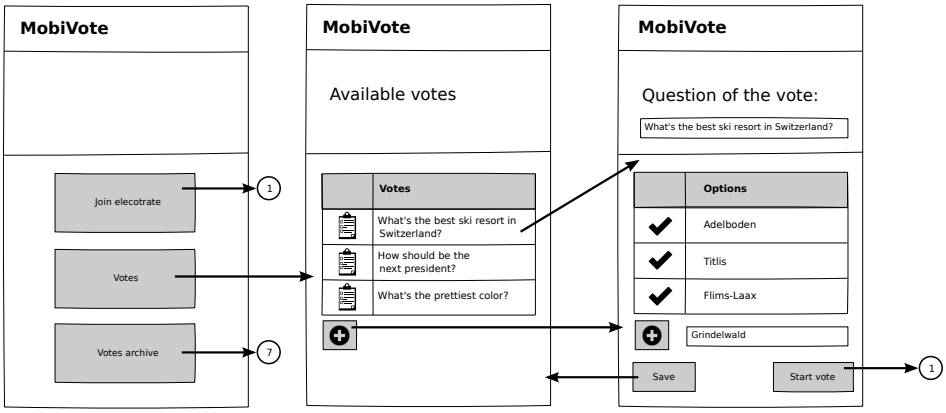

Figure 4.1: Setup a vote

Network Setup. The next step after defining a vote is to define in which network environment the voting session will take place. The first person to do so is the vote administrator. He can choose a Wi-Fi network in which the network session will be started. The information about the chosen network as well as the generated session

password has to be shared with the participants. To do so, the administrator can either just read the information out loud, or share the information using either a QR-code or write the information to a NFC token which can be passed to all participants. Using this information, the participants can join the network session with their devices. Before joining the session, each participant (including the administrator) has to define an identification string for themselves. The screen design of this phase is illustrated in Figure 4.2.

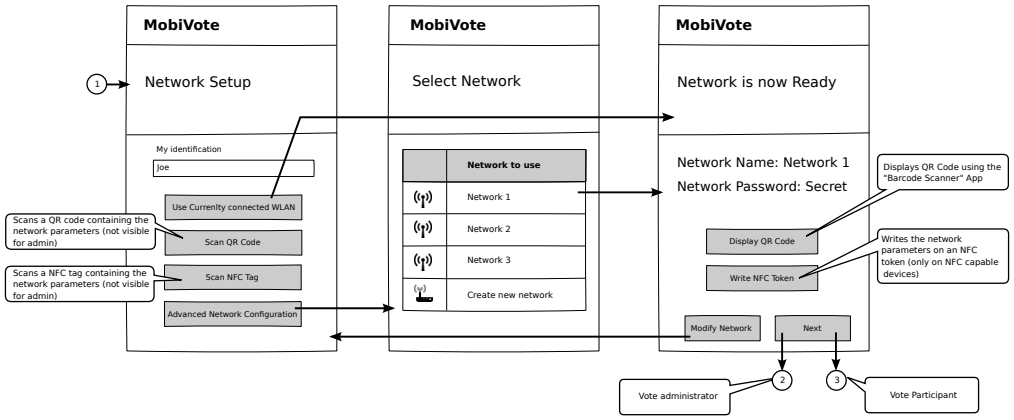

Figure 4.2: Network setup

Defining the Electorate and Review of the Vote. After joining the network session, a list of all the session participants is displayed. At this stage, the administrator can define the electorate, meaning he can select the participants which are allowed to vote. This can be done by clicking on the checkboxes next to the participant's identification. The participants can see who has been selected on their devices.

Once the administrator has defined the electorate, each participant has to confirm that all the parameters of the vote (question, allowed options and electorate) are correct. The vote can only be started by the administrator if the whole electorate has agreed. If necessary, the administrator can go back and do changes on the vote parameters.

The screen design of this phase from the administrator's perspective is illustrated in Figure 4.3 on the next page and from the voter's perspective in Figure 4.4 on page 36.

Voting phase. Once the vote has passed the review, the vote itself can take place. Each participant can choose an option and cast it. The screen transitions to a view which shows the vote casting state of all the participants. The screen design of this phase is illustrated in Figure 4.5 on page 36.

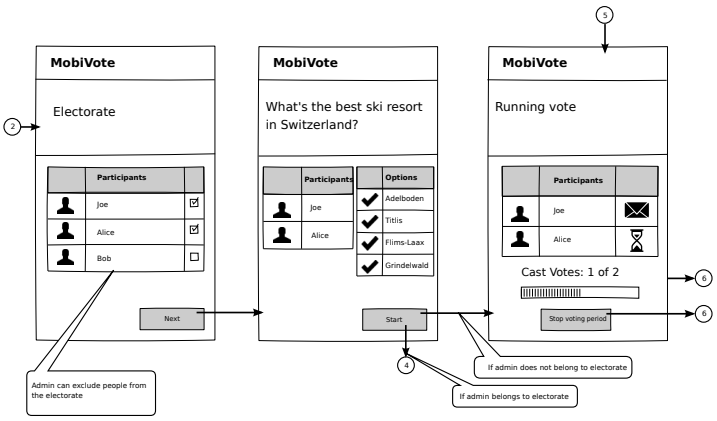

Figure 4.3: Administrator defining the electorate

Displaying the result. The vote is tallied as soon as the last vote has been cast, or if the administrator has ended the voting phase. The result is displayed in form of a pie-chart and a tabular view. The result of the vote is saved on each device and can be revisited using the archive function in the main screen. The screen design of this phase is illustrated in Figure 4.6 on page 37.

4.2.2 Implementation

The user interface has been implemented according to the storyboard presented in the previous section. The Android platform provides a powerful framework to implement rich user interfaces. The framework also allows to handle different layouts for each individual form factor. This is necessary because devices can have different screen sizes and are normally used in different orientations. A layout designed for a smartphone in portrait mode usually does not fit well on a tablet in landscape mode. To deal with this issue, multiple layouts have been developed to cope with this issue. The Android framework then automatically applies the correct layout.

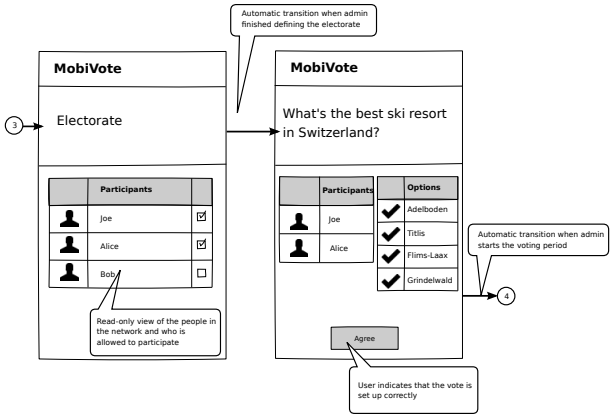

Figure 4.4: Review of the vote from voter's perspective

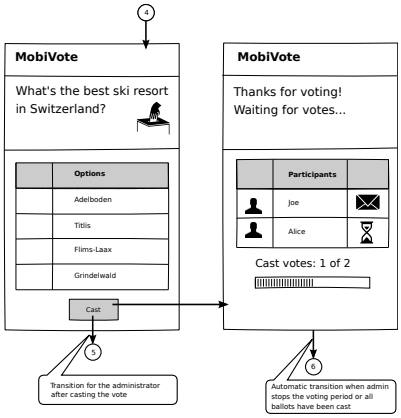

Figure 4.5: Vote casting

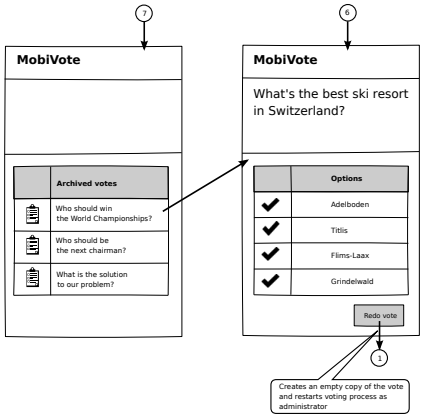

Figure 4.6: Result of the vote

4.3 Bootstrapping of Group Communication

In order to assure privacy and authenticity of a voting session, the communication between the participant's devices needs to be secured using cryptographic means. To achieve that, different types of cryptographic keys need to be exchanged. When joining a voting session, the user needs to provide a conversation password which has been generated on the device of the vote administrator. The administrator has several possibilities to distribute this password, along with the other information which is needed to connect, namely the SSID of the WLAN on which the conversation takes place and the group name to identify the conversation. The following possibilities are available:

- Communicate the password, the group name and the SSID as text. Participants who want to join the session need to enter these parameters using the soft keyboard of the mobile device.
- Display a QR-code which encodes password, group name and SSID. Participants who want to join the session can snap a picture of this QR-code using the camera of the mobile device and decode the information on their devices.
- Write password, group name and SSID to a NFC tag which then can be passed along to all the participants. When tapping this NFC tag on the back of a NFC capable device, the User will be connected to the session.

The application uses the password to derive an AES key which will be used to symmetrically encrypt all the broadcast traffic in the conversation. Furthermore, a dynamic salt which is generated by the administrator decreases the risk of rainbow table attacks.

In order to guarantee the authenticity and integrity of the messages in the conversation, each participant generates an RSA key pair. The public key is broadcast to all the participants. This key setup can then be used for signing/verifying all the messages in the conversation, as well as for establishing a secure point-to-point channel between two participants in the conversation.

For the unicast functionality of the messaging layer, a hybrid encryption mechanism is used. To achieve this, we encrypt an AES key which is used previously to encrypt the unicast payload using the RSA public key of the recipient. The recipient can then use his private key to decrypt the AES key, and then in a second step use this AES key to decrypt the payload.

4.4 Messaging

For the communication between devices and components, a messaging infrastructure is required. We can divide the messages which are exchanged during a voting period into two categories:

- **Network Messages:** Messages which are exchanged between the participant's mobile devices.
- **Local Broadcast Messages:** This type of messages are Messages which are passed between Android components, for example: A user interface component needs to be updated if a specific event occurs.

In order to distinguish the messages in order to decide how they should be processed, messages types were defined. The following two sections describe the message types which have been defined for each category and provide further technical insight how the messaging infrastructure has been designed.

4.4.1 Network Messages

Network Messages are required to exchange messages between the mobile devices of the participants. This is necessary for establishing the voting session, as well as for exchanging protocol specific values, for example an encrypted ballot. These messages are passed through a stack of communication layers. Figure 4.7 on the next page depicts how this stack is built. For establishing a group communication among network devices, the AllJoyn library is used. For the specific use in MobiVote, a wrapper library has been implemented. The availability of this interface allows to interchange the messaging library which is used for MobiVote. The AllJoyn interface library allows to send broadcast messages which reach all the participants of the group. Most of the communication is done in this manner. It is also possible to send a message to a specific member of the group. This method is called unicast. In order to distinguish the different purpose of the messages interchanged between the participant's devices, a number of message types have been defined. These message types are outlined in Table 4.1 on page 42.

4.4.2 Local Broadcast Messages

The Android framework provides an internal messaging infrastructure which allows Android components to communicate among each other. Any component can broadcast messages using this infrastructure. These messages can be received by any component which registered a so called Broadcast Receiver. All the registered components will then

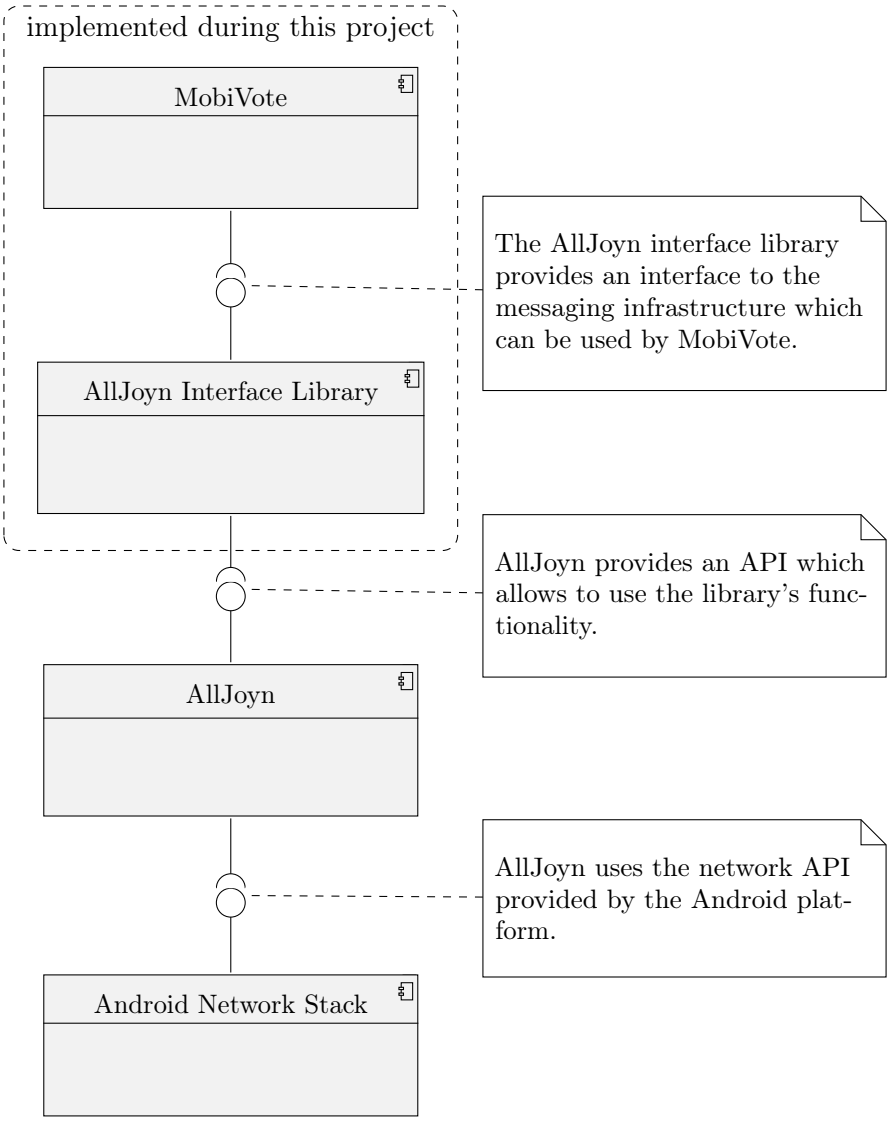

Figure 4.7: Messaging architecture

be notified as soon as a message has been broadcast. This mechanism can be used for example to update the User Interface according to a specific event. This paradigm was used heavily for the implementation of this project.

Message Type	Description
VOTE_MESSAGE_ELECTORATE	Used during the setup phase to indicate which participants are allowed to vote.
VOTE_MESSAGE_POLL_TO_REVIEW	Used to signal the participants that a vote is available for review.
VOTE_MESSAGE_ACCEPT_REVIEW	Used by the participants to signal that they agree with the proposed vote.
VOTE_MESSAGE_START_POLL	Used to signal the start of a voting period.
VOTE_MESSAGE_COEFFICIENT_COMMITMENT	Used during the key generation phase for committing to the chosen coefficients.
VOTE_MESSAGE_KEY_SHARE	Used to transmit the key shares between participants.
VOTE_MESSAGE_KEY_SHARE_COMMITMENT	Used to commit to the key share used for performing the part decryption.
VOTE_MESSAGE_VOTE	Used to transmit a vote.
VOTE_MESSAGE_STOP_POLL	Used to signal the end of a voting period.
VOTE_MESSAGE_CANCEL_POLL	Used to signal the cancellation of a voting period.
VOTE_MESSAGE_PART_DECRYPTION	Used to transmit a part decryption after the voting period for jointly decrypting the result of the vote.

Table 4.1: Network message types

4.5 Data Structures

In this section the used data structures to store data within the MobiVote application are introduced.

Database. For the management of the available votes in the system (prepared as well as completed votes), a small database structure has been implemented. The Android platform embeds the sqlite[9] database in its API, hence this was the obvious choice for the database technology. Figure 4.8 depicts the very simple data structure which has been chosen for this project.

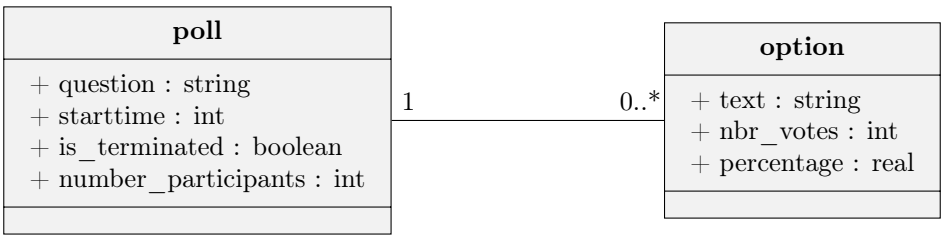

Figure 4.8: Entity relationship diagram

According to the Android best practice, a `SQLiteOpenHelper` class has been implemented which manages all the database communication. It also takes care of the mapping between entity objects and database entries. Figure 4.9 on the following page depicts the implemented entity classes. The entity class which represents a participant does not have a corresponding database identity, this is because the participants of the vote are defined during the voting cycle. It is also not relevant later on when displaying the result of the vote, therefore the information about the participants is not stored in the database.

XML Serialization. In order to make a vote verifiable, all the relevant values such as cryptographic parameters, encrypted ballots, zero-knowledge proofs, etc. of the CGS97 voting protocol are serialized into a simple XML file. To do so, we are using the Simple framework. Using this framework, we can create a set of classes which are used to store all the values which should be written into a file. Instructions how the serialization should happen are defined using annotations in the class. This type of simple class is also known as POJO (Plain Old Java Object)[10]. These POJO classes could later be used for writing a

[9] `http://www.sqlite.org/`
[10] `http://en.wikipedia.org/wiki/Plain_Old_Java_Object`

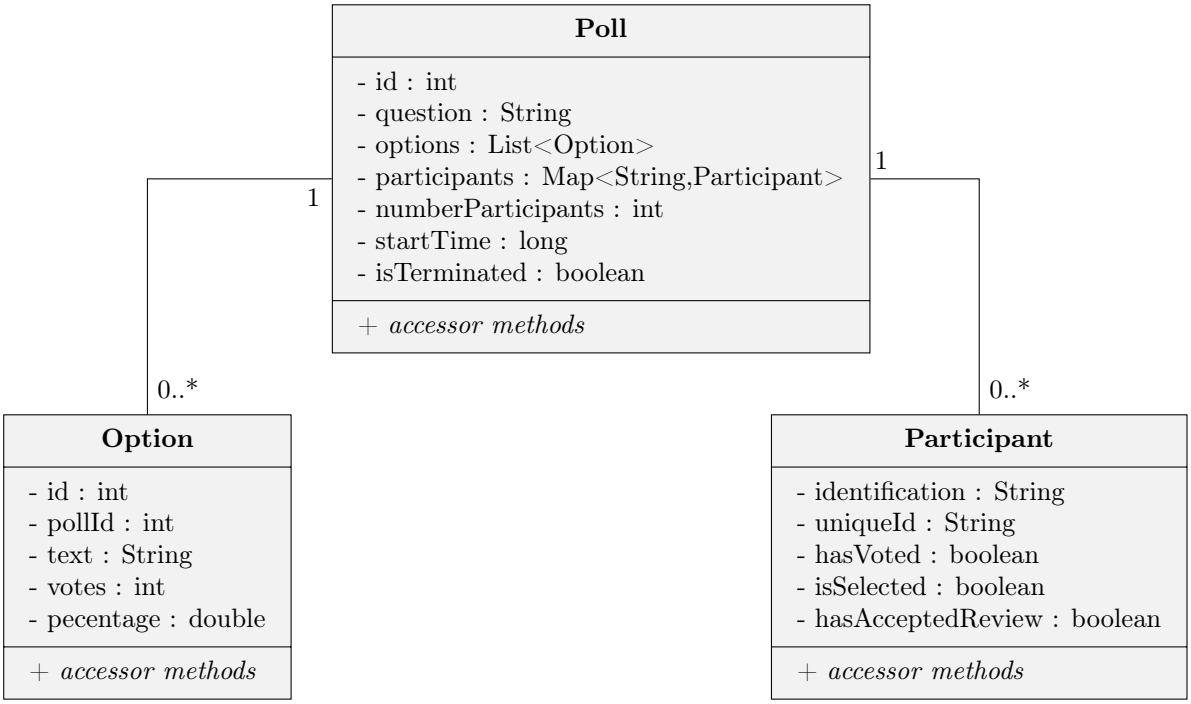

Figure 4.9: Entity class diagram

verifier software which allows to check a vote by a third party software. The POJO classes can be found in the Java package `ch.bfh.evoting.voterapp.protocol.cgs97.xml`.

4.6 Voting Protocol

A major goal of this project is to create a decentralized e-voting system which provides as many security properties as possible. These properties are discussed in Section 2.1 on page 5. The CGS97 voting scheme [3] offers most of these properties. The properties of the CGS97 scheme are discussed in Section 2.3 on page 16. This section gives an overview of some aspects which have been considered during the implementation.

4.6.1 Modifications in the Protocol

The CGS97 scheme has not specifically been designed for decentralized e-voting systems. The protocol itself remains exactly the same, but the assignment of some roles are slightly changed:

Trustees: The CGS97 scheme stipulates the role of trustees. Trustees are responsible to jointly generate an asymmetric key pair for the voting session and jointly decrypt the result after the voting period has come to an end. In the decentralized scenario, we merge the role of the trustee with the role of the participant. This does not create a conflict because a trustee can also be a participant in the vote. The fact that we assign the role of a trustee to each participant also reduces the complexity from a usability perspective, because no explicit assignment is required during the setup process of the vote.

Bulletin Board: In classic e-voting scenarios, a bulletin board is considered as a centralized high available infrastructure. This paradigm cannot be applied to the decentralized voting scenario because no central infrastructure should be required. Instead, each participant creates its own transcript of the voting protocol. Almost all messages are exchanged by broadcasting them to all participants of the group and can be intercepted and recorded by each participant. In case of a manipulation, the participants would discover the manipulation by comparing the recorded values using their physical proximity.

4.6.2 Ballot Encoding

One goal of this project was to build an e-voting system that supports *one-out-of-k* voting scenarios. This functionality can be achieved by employing a ballot encoding mechanism. We discussed two possible vote encodings in Section 2.4 on page 20 we discussed two

approaches, each with its own advantages and disadvantages. The single encryption ballot encoding is the more efficient approach in the voting phase. In the tallying phase on the other hand, this approach becomes very expensive as the number of options and participants increase. At this point, the multi encryption ballot encoding comes in. In order to get better performance in small voting scenarios while being able to perform votes with a big number of votes and participants, a hybrid system has been implemented. Using the strategy pattern, the protocols can be interchanged. The decision which strategy is going to be used is reached by calculating the number of steps which are required to tally the vote in a single encryption ballot scenario. In order to calculate the number of these steps, the following formula can be used, k being the number of options available, n the number of voting participants and s the resulting number of steps:

$$s = \binom{k+n-1}{k-1}$$

Further background regarding this relation can be found in [1] and [7]. As a threshold value for s, the number 10 000 has been chosen. This value seemed to be a reasonable balance, considering that the evaluation of a combination takes roughly 1.8 milliseconds on a modern device. That means that we allow a tallying time of 18 seconds on modern devices. If s is smaller, the single encryption ballot encoding is used, the multi encryption ballot strategy otherwise.

4.6.3 Vote Export

We have seen that the CGS97 protocol is verifiable. The implemented application of course does all the checks in order to make sure that nothing goes wrong during the voting process. For the user, these security mechanisms are not visible, and therefore a user could question the soundness of the voting process. Since all the relevant cryptographic values are recorded and saved, we can export these values so that a third party verification software could verify the voting process. This export functionality exports all the values of a vote into a XML File. The XML structure depends on the vote encoding that has been chosen. Figure 4.10 on the following page depicts the structure which is used for a multi encryption ballot vote, Figure 4.11 on the next page the structure used for a single encryption ballot vote.

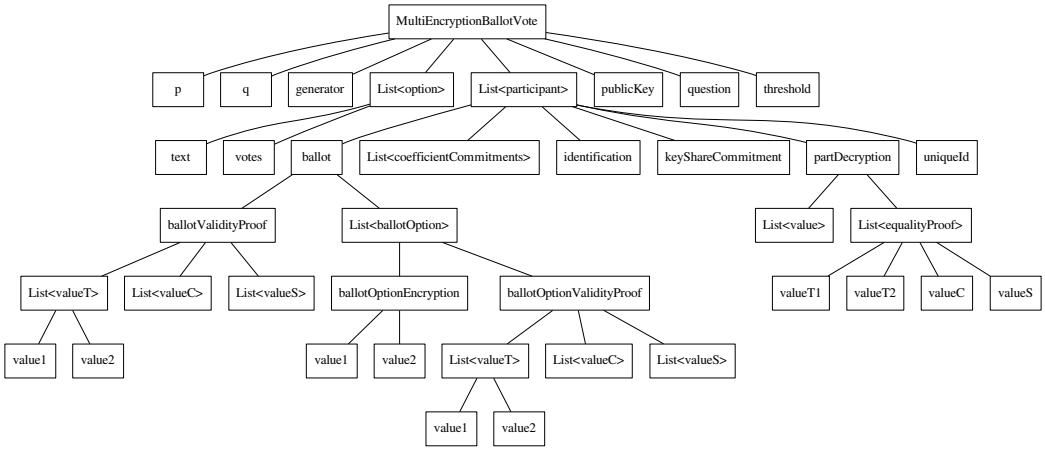

Figure 4.10: XML structure of a multi encryption ballot vote

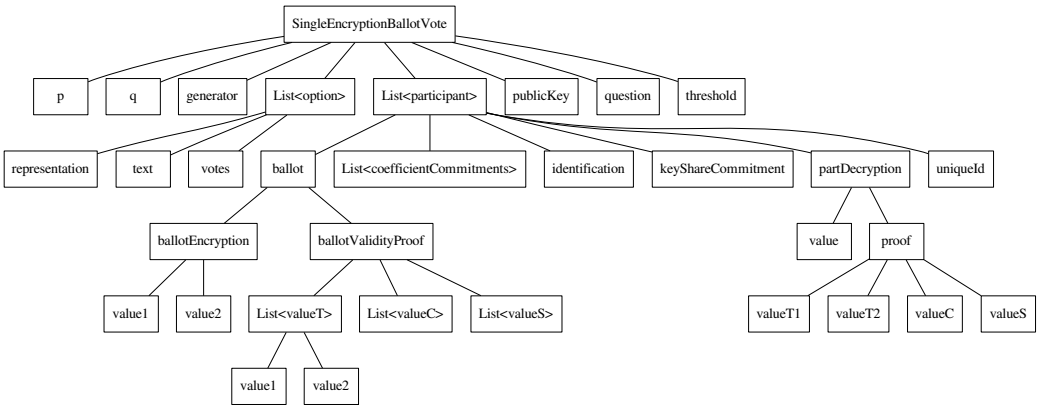

Figure 4.11: XML structure of a single encryption ballot vote

4.7 Testing

To verify the correct behaviour of the implemented product, a number of different tests have been conducted during the implementation phase. These testing activities are outlined in the following.

4.7.1 Functionality Tests

The functionality tests took place right after the implementation step. They were all done manually on the available devices. The devices outlined in Table 4.2 were available for testing the functionality.

Device	Form factor	Android Version
Samsung Galaxy S 3	Smartphone	4.3 (Samsung original)
Samsung Galaxy S 1	Smartphone	4.3.1 (CyanogenMod)[a]
Samsung Galaxy Tab 10.1	Tablet	4.0.4 (Samsung original)

[a] http://www.cyanogenmod.org/

Table 4.2: Testing infrastructure

4.7.2 Usability Tests

The implementation phase of this project started with the implementation of the Graphical User Interface of the application. At this stage, it was possible to setup and run a vote on multiple devices, but without the security of the later implemented e-voting protocol. This implementation was then used to verify that the user interface of the application does not cause any confusion among users. Several people without any background in the e-voting area were asked to use the application and provide feedback. The feedback we got from the first implementation showed potential for optimization, details such as the placement of buttons or the purpose of some user interface elements. The overall feedback showed, that we were able to build a e-voting application which is easy to use. The implementation of the e-voting protocol which was done after the usability tests mostly took place behind the scenes, there were only some slight modifications in the user interface necessary to add this functionality.

4.7.3 Load Tests

In a decentralized environment, the load which each device has to handle grows with the number of participants and the number of options in a vote. In order to verify that the implementation is able to handle scenarios with more than just 3 participants, 17 LG Nexus 5 smartphones running Android version 4.4 were available to test more complex scenarios. Various tests with different scenarios have been performed and assessed regarding time consumption in the two most time consuming steps, namely casting a ballot tallying a vote. The results of these performance tests are shown in Table 4.3. The performance tests were executed using the two separate ballot encoding strategies as discussed in Section 2.4 on page 20. Another relevant factor regarding performance is the bit length of the cryptographic parameters p and q. For all tests, parameters with the length of 2048 bits have been used.

		Single Ballot Encryption		**Multi Ballot Encryption**	
Options	**Participants**	**Cast Ballot**	**Tallying**	**Cast Ballot**	**Tallying**
4	3	2.0 s	1.2 s	3.7 s	3.7 s
4	4	2.0 s	1.3 s	3.4 s	3.6 s
4	5	2.0 s	1.3 s	3.2 s	4.0 s
6	3	2.6 s	1.2 s	4.3 s	4.8 s
6	4	2.4 s	1.5 s	4.8 s	5.4 s
6	17	2.5 s	39.0 s	4.6 s	22.0 s
10	17	2.4 s	-	6.7 s	47.2 s

Table 4.3: Performance test results

These results clearly show that the single encryption ballot strategy is the better choice for small voting scenarios, looses its advantages in bigger scenarios because the number of combinations which have to be checked during the tallying process grows factorially. One combination check (which is basically a modular exponentiation) was measured to take around 1.8 milliseconds on a LG Nexus 5 smartphone. In the scenario with 10 options and 17 participants, the tallying process would take roughly 2 hours and 40 minutes, that is why there is no result for this scenario in the table. Using the multi encryption ballot

strategy on the other hand, tallying the same scenario takes 42 seconds.

The performance tests were executed on modern smartphone hardware which can perform expensive operations such as generating encryptions and proofs in a reasonable time. The user experience on older devices such as the Samsung Galaxy S1 offer a less smooth user experience, especially when using the multi encryption ballot strategy which requires much more calculation power for creating and verifying ballots.

The load tests with many devices also unveiled another fact: The AllJoyn communication framework only allows 16 devices to join into a communication group. This restricts the voting scenario to a maximum of 17 participants, including the administrator who initiated the communication session. According to the documentation of AllJoyn, the library does not have such a restriction, which after a glance into the source code of AllJoyn proved to be not true. This is why this restriction has been discovered only in a late stage of the project.

4.8 UniCrypt Extensions

The CGS97 protocol contains a few building blocks which were not part of the UniCrypt library by the time this project started. Part of the goals of this project was to enhance UniCrypt so that the logic could be used in the implementation of this application and make it also reusable in further upcoming projects. In the following we describe how this implementation has been done.

4.8.1 Shamir Secret Sharing Scheme

In 1979, A. Shamir proposed a scheme [18] on how a secret can be split into several independent pieces which can be distributed among several authorities. It is not possible to recover the secret unless a sufficient amount of these authorities collaborate in order to recover the secret. Further details regarding this scheme can be found in Section 2.2.4 on page 9.

Implementation. The implementation has been embedded into the existing UniCrypt structure. The class in which the implementation has been done is the following:
`ch.bfh.unicrypt.crypto.schemes.sharing.classes.ShamirSecretSharingScheme`

The instantiation of this scheme requires three parameters:

- An algebraic field of the type `ch.bfh.unicrypt.math.algebra.dualistic.classes.ZModPrime`. The secret which

should be shared later on has to be an element of this field.

- The number of shares which should be produced
- A threshold value which defines how many shares must be reunited in order to recover the secret

A secret can be shared using the method `share()`, which takes a message as an argument. The message must be an element of the field which has been specified during the initialization of the scheme. The method returns an array of value pairs. Each pair represents a share.

The secret can be recovered using the method `recover()`, which takes an array of value pairs as an argument. The array must at least contain as many value pairs as specified with the threshold value during instantiation. The method returns a single element which represents the recovered secret.

Testing. UniCrypt components are tested using the JUnit[11] framework. A JUnit Test for the Shamir Secret Sharing scheme has been implemented in order to check the correct functionality after each change that is done in the UniCrypt library. The following test class has been implemented:

`ch.bfh.unicrypt.crypto.schemes.sharing.ShamirSecretSharingSchemeTest`

4.8.2 ElGamal Proof of Validity

A proof of validity is required if a party wants to prove that an ElGamal cipher text is the encryption of element of a set of possible plain texts without revealing which one it is. Section 2.2.5 on page 14 covers this topic in further detail.

The implementation of the this proof has been done in a very early stage of this project, according to the architecture of UniCrypt 1. In the meantime, UniCrypt has been completely re-engineered. The validity proof has been re-implemented independently of this project in order to fit into the new architecture. Since this project is based on UniCrypt 2, the project makes use of the re-implemented version of the ElGamal proof of validity.

[11] `http://junit.org/`

5 Discussion

At the end of this project, we reached the goal of creating an e-voting application which runs on mobile devices and can be used without additional infrastructure and offers security features such as privacy, anonymity and verifiability. This chapter reflects on certain aspects of this work.

5.1 Decentralized E-Voting

This section examines the term *decentralized* in relation to this project and what approaches have been chosen to achieve decentralized e-voting.

Communication Layer: During this project we implemented an e-voting system which does not require any other infrastructure than the devices of the participants (smartphone or tablets). As a communication layer for the communication between the devices, WLAN is used. Many facilities provide a WLAN infrastructure. The application provides the functionality to use this infrastructure, but also offers the possibility to set up a new WLAN network using the hotspot functionality of Android. The hotspot integrated in older devices is usually not as powerful as a hotspot which is designed for large networks. For larger scenarios it is recommended to use a proper infrastructure WLAN hotspot as the communication layer. One can argue that this violates the principle of decentralization. This is true for the base communication layer, but it does not affect the decentralized behaviour of the e-voting protocol which is running on top of this communication layer.

Messaging Layer: In a preceding project, a decentralized messaging infrastructure for Android called InstaCircle [14] has been implemented. The goal is to establish a group communication so that the participating devices can exchange messages among each other. The message exchange in InstaCircle is based on broadcasts in an IP network. Because broadcast communication does not offer a reliable message exchange, this approach turned out to be not reliable enough in certain network situations. The messaging layer has been replaced by the AllJoyn library (see Section 4.1.2 on page 31). In contrast to InstaCircle, the initiator of a group takes the role as a master who orchestrates the communication

within the group. In some sense, this violates the principle of decentralization too. The choice of this architecture can be justified for two reasons though:

- The master only orchestrates the message exchange. Attempts to cheat on the e-voting protocol layer would be recognized by the protocol.
- In a voting session, some sort of administrator is necessary. This person is predestined to take the master of the group communication as well.

Regardless of who manages the group communication, from an infrastructure point of view, the communication between the devices can still be done in a decentralized manner.

E-Voting Protocol: As we have seen in Section 2.3 on page 16, the protocol has not specifically been designed to be used in decentralized scenarios. The protocol has been adapted in two major points in order to achieve this property:

- All the participants become trustees and contribute to the key generation and decryption process
- All participants have their own bulletin boards

More details regarding these modifications can be found in Section 4.6 on page 45. The fact that there is no central bulletin board also means that there is no master on which the final result is derived from. Even though several security measures have been built into the application, we still have to consider that this software runs on devices which are potentially compromised. For example, a malware could fake the screens when the application asks for input or displays the result. At this point, we can take advantage of the fact that the votes only take place in a confined space where the participants can communicate visually and orally without using their devices. In the case of a disagreement on the official result of the vote, the participants can compare the result displayed on their device and then take one of the following actions:

- Agree on the result which is displayed on the majority of devices.
- Rerun the vote, potentially with different devices.

Using this method of negotiating without using the devices should allow to agree on an official result of the vote.

Verifiability: Using the recorded values on the bulletin board, the whole voting cycle can be verified. It stands for itself that the MobiVote application verifies all the values, but in order to increase the confidence in MobiVote, it would be desirable that a vote can be verified by an external software. To do so, MobiVote offers the functionality to export the values of the bulletin board to an XML file, which then can be used as an input for an external verifier software. One missing link in this verifiability chain stems from the fact that the identities used in a vote are created ad-hoc when establishing the group communication before voting. That means that the integrity of the messages which are exchanged can only be verified during the session. Of course, the public keys of all participants used during a particular voting session could also be exported, but could be replaced along with all the signatures which are recalculated using the private key corresponding to the new public key. This missing link could be added by using identities verified by a trusted authority, for example with digital certificates. Dealing with certificates usually has a significant negative impact on the usability. Furthermore, only a minority of people are in possession of such a digital certificate which could be used as a digital identity during the voting process.

5.2 Usability

In retrospective, the decision to create a proper storyboard before we started to implement the user interface was a very good decision. It forced to reflect on how such a spontaneous e-voting process should look like in practice. It turns out that the process in such a spontaneous manner looks different compared to a classic e-voting scenario in many ways. For example, we need to find a way on how the electorate and setup can be defined and, more importantly, approved by everybody before the vote itself can start. When we consider classic voting scenarios, the electorate is clearly defined in advance, for example all eligible citizen of a country. The same is true for defining how the question and the possible options should be defined. Quite some paperwork usually precedes a vote, whereas in a spontaneous vote, the setup needs to be approved somehow.

Before implementing the actual e-voting protocol which provides the e-voting specific security measures, we had an implementation ready which allowed to perform a vote in a decentralized manner, but did not offer the desired security. We used this stage to do tests of the communication layer and, more importantly usability tests (see Section 4.7.2 on page 48). The fact that the user interface at this stage and at the final stage almost looks identical shows two things:

- We were able to implement the e-voting protocol in an almost transparent manner.

- The storyboard that has been designed seems to be universal and not specific to this implementation.

The latter fact is also underpinned by the fact that the partner project came to the same conclusion (see Section 5.4 on the next page).

5.3 Security Considerations

It speaks for itself that security plays a crucial role in the context of e-voting. In this project we employed modern cryptographic techniques which are the answers to many threats. Nevertheless we had to make some compromises. Most of them relate to the tradeoff between security and usability. This section should give an overview of these compromises.

Public Key Exchange: In order to guarantee the authenticity of messages which have been exchanged during a voting session, each participant needs a private key for signing all the outgoing messages. All the other participants need to be in possession of the corresponding public key, which they use to verify incoming messages of that particular sender. Obviously these public keys have to be exchanged at the beginning of the session. Of course, these keys can be transmitted using the existing network communication layer, but since these keys are not linked to a real life identity, we can only assume that the key really belongs to the sender. During this key exchange process, a participant could hence impersonate somebody else. The key exchange happens on the secured layer, so that impersonation would have to be done by a participant who is in possession of the group password. This fact reduces the risk significantly, because since all participants are gathered in the same physical space, the fact that something is wrong would be discovered. The fact that non-certified keys are used for each session also makes it impossible to verify the authenticity of the messages after the voting session.

This problem could be solved if each participant has its own certificate, which is a certified digital identity that lasts for longer than just a voting session. Unfortunately only a small percentage of people possess a digital certificate nowadays. Requiring such a certificate would narrow the group of potential users further down and would have a negative impact on the security.

Untrusted Devices: Nowadays, smartphones are little computers which allow to run all sort of software. The ever growing popularity of these devices also caused malware to appear. Malware could be employed to circumvent security measures, for example by showing a user a fake screen on its devices. For example, during a vote with options A

and B, the malware would display option B instead of A and vice versa. This causes a voter that intends to vote for option A to vote for option B. Malware could also be used to collect information on how a participant voted. Dealing with untrusted platforms is subject of ongoing research and is out of scope of this project.

5.4 Comparison to the Partner Project

During the same period, a project with the same focus, but with a different e-voting protocol has been implemented [1] (see Chapter 3 on page 26). This allows us to compare each others implementation. In both projects, the specified e-voting protocol could be implemented successfully. Also, both projects share the same user interface, which shows that the storyboard is generic. There is only one small deviation in the implementation of this project, namely the threshold parameter which has to be defined in the secret sharing part of the CGS97 e-voting protocol. Sharing the same user interface also means that the protocol implementation could be merged into a single app and let the user choose which protocol to use.

As we found out, both protocols have their own advantages and disadvantages, mainly in the area of performance. The HKRS12 protocol [10] which has been implemented in the partner project is specifically designed for this kind of voting scenarios. It is also a very slim protocol which results in a faster execution, especially on slower devices. A disadvantage is the fact, that the decoding step of the HKRS12 protocol requires an exhaustive search of combinations in order to avoid the discrete logarithm problem, very similar to the single encryption ballot encoding used in this project (see Section 2.4.2 on page 21). This means that the time consumption of the tallying process grows rapidly as the number of participants and options increase. In this implementation we have the same problem when using the single encryption ballot strategy, but in this implementation we have the option to switch to the multi encryption ballot strategy (see Section 2.4.1 on page 20).

5.5 Challenges

Along the way to the finished app, there were a few challenges to overcome. This section should give an overview on some points that turned out to be a bit more challenging than anticipated.

Decentralized Communication: The original plan was to employ the InstaCircle communication infrastructure in the project. InstaCircle is a completely decentralized

communication infrastructure. That also means that here is no master instance which manages the overall state of the communication session. Problems with synchronisation and reliability arise. This is one reason that the communication layer has been switched to the AllJoyn library.

Security - Usability Trade-off: From a user's point of view, security usually puts many obstacles in the way. PIN-codes, passwords, keys, certificates and similar things are confusing the users. To keep the usability on an acceptable level, some compromises have been made. One such compromise is the use of non-certified identities. Of course, the design of a usable user interface was quite a challenge on its own.

Android Wi-Fi API: The android platform provides an API for managing the Wireless connections on an Android device. This API is used in our application to properly configure the network on which the voting session is going to take place. Despite the documentation, it took quite some time to figure out how this API can be used properly.

Robustness in many Situations: An application running on a mobile device has to be in a way that it can deal with many unexpected situations. For example, an incoming phone call should not disrupt the entire ongoing voting process. In a voting scenario with many participants, the chance that something unexpected happens are quite high. If these situations are not handled properly, the votes have be rerun many times until everything works as expected.

Ongoing Development of UniCrypt: During the implementation phase of this project, UniCrypt was still under heavy development and was undergoing a lot of changes, also on the exposed interfaces on the library. Updating to the newest snapshot version of UniCrypt often also required changes in the implementation of the protocol.

6 Conclusion

During this project, we implemented a mobile e-voting application which can be used for boardroom voting. It is running on the popular mobile platform Android and therefore be used on devices such as smartphones and tablets. There is no infrastructure other than the participant's devices required in order to run a vote on them. During this project, we also contributed to the UniCrypt library and implemented cryptographic building blocks which were used to implement the application at hand. We were also able to give feedback regarding the ongoing implementation of UniCrypt and also found some bugs along the way. Of course, these bugs have been fixed by the UniCrypt development team in the meanwhile. The fact that we could implement multiple cryptographic protocols relatively easy also proves, that UniCrypt is heading in the direction that it was intended to.

The user interface design of a decentralized e-voting system requires some additional considerations, for example there is a mechanism required which allows the participant to approve a certain vote setup. We invested a good deal of time into building a user interface for mobile devices that allows to deal with this additional requirements. After some unsatisfactory attempts, we ended up with a design that met the requirements, proved by usability tests. The user interface has been designed in cooperation with another master's thesis project [1]. This cooperation proved to be extremely valuable, thanks to a lot of discussions and different perspectives. The fact that the resulting user interface was also used in the master's thesis project of Philémon von Bergen shows that we built a truly universal user interface for decentralized e-voting systems.

6.1 Future Work

The application at hand could be extended in many different ways. In this section we present some ideas for extensions which could be realized in the future.

Mobile Platforms: The current implementation is limited to devices running the Android mobile operating system. All participants have to be in possession of an Android device. Despite the fact that Android is an extremely popular platform, this would be a big coincidence. To enable users of other mobile platforms to participate on votes, this

MobiVote implementation should be re-implemented on other platforms such as Apple iOS or Windows Mobile.

Vote Verifier: The MobiVote application allows to export all relevant cryptographic values which are required to verify the whole vote process. After exporting these values, they could be fed into a verification software that recalculates all necessary steps and recalculates potential problems.

Identity Management: At this point, MobiVote works with temporary identities for each voting session. This fact makes it impossible to check the integrity of messages after the voting session ended because the identity cannot be assigned to a human participant or at least to a device. This issue could be solved if MobiVote could use identity providers to assign a participant distinctively to a identity.

Multiple Protocols: Another interesting experiment would be to merge the application implemented in this project and in the partner project by Philémon von Bergen [1]. An application featuring multiple e-voting protocols could choose automatically the most efficient protocol according to a scenario (e.g. number of participants or number of options), or even give the user the choice which protocol should be used for a voting session.

This is a non-exhaustive list of proposals for following projects, but it clearly shows that the research in the field of decentralized e-voting is certainly not at its end and there is still a lot of room for interesting projects in this area.

Bibliography

[1] Philémon von Bergen. "A Mobile Application for Boardroom Voting". MA thesis. Bern University of Applied Sciences, Engineering and Information Technology, Feb. 2014. URL: http://e-voting.bfh.ch/students/theses-reports.

[2] David Chaum and Torben Pryds Pedersen. "Wallet Databases with Observers". In: *Advances in Cryptology — CRYPTO' 92*. Vol. 740. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1993, pp. 89–105. ISBN: 978-3-540-57340-1. DOI: 10.1007/3-540-48071-4_7. URL: http://dx.doi.org/10.1007/3-540-48071-4_7.

[3] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. "A Secure and Optimally Efficient Multi-Authority Election Scheme". In: *Advances in Cryptology — EUROCRYPT '97*. Vol. 1233. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, pp. 103–118. ISBN: 978-3-540-62975-7. DOI: 10.1007/3-540-69053-0_9. URL: http://dx.doi.org/10.1007/3-540-69053-0_9.

[4] Taher El Gamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *Information Theory, IEEE Transactions on* 31.4 (1985), pp. 469–472. ISSN: 0018-9448. DOI: 10.1109/TIT.1985.1057074. URL: http://doi.acm.org/10.1109/TIT.1985.1057074.

[5] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Vol. 263. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1987, pp. 186–194. ISBN: 978-3-540-18047-0. DOI: 10.1007/3-540-47721-7_12. URL: http://dx.doi.org/10.1007/3-540-47721-7_12.

[6] Rolf Haenni and Oliver 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*. EVT/WOTE'11. San Francisco, CA: USENIX Association, 2011, pp. 8–8. URL: http://dl.acm.org/citation.cfm?id=2028012.2028020.

[7] Feng Hao, Peter Y. A. Ryan, and Piotr Zielinski. "Anonymous voting by two-round public discussion". In: *IET Information Security* 4.2 (2010), pp. 62–67. DOI: 10.1049/iet-ifs.2008.0127. URL: http://dx.doi.org/10.1049/iet-ifs.2008.0127.

[8] James Heather and David Lundin. "The Append-Only Web Bulletin Board". In: *Formal Aspects in Security and Trust*. Vol. 5491. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 242–256. ISBN: 978-3-642-01464-2. DOI: 10.1007/978-3-642-01465-9_16. URL: http://dx.doi.org/10.1007/978-3-642-01465-9_16.

[9] Hugo Lennaert Jonker. "Security Matters: Privacy in Voting and Fairness in Digital Exchange". PhD thesis. Eindhoven University of Technology and University of Luxembourg, 2009.

[10] Dalia Khader et al. "A Fair and Robust Voting System by Broadcast". In: *Electronic Voting*. Ed. by Manuel J. Kripp, Melanie Volkamer, and Rüdiger Grimm. Vol. 205. LNI. GI, 2012, pp. 285–299. ISBN: 978-3-88579-299-4.

[11] Ueli Maurer. "Unifying Zero-Knowledge Proofs of Knowledge". In: *Progress in Cryptology – AFRICACRYPT 2009*. Vol. 5580. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 272–286. ISBN: 978-3-642-02383-5. DOI: 10.1007/978-3-642-02384-2_17. URL: http://dx.doi.org/10.1007/978-3-642-02384-2_17.

[12] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996. ISBN: 0-8493-8523-7. URL: http://cacr.uwaterloo.ca/hac.

[13] Torben Pryds Pedersen. "A Threshold Cryptosystem without a Trusted Party". In: *Advances in Cryptology — EUROCRYPT '91*. Vol. 547. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1991, pp. 522–526. ISBN: 978-3-540-54620-7. DOI: 10.1007/3-540-46416-6_47. URL: http://dx.doi.org/10.1007/3-540-46416-6_47.

[14] Jürg Ritter. *InstaCircle – A Location-Bound Ad-Hoc Network for Android Devices*. Tech. rep. Bern University of Applied Sciences, Engineering and Information Technology, Feb. 2013. URL: http://e-voting.bfh.ch/students/theses-reports.

[15] Jürg Ritter. *UniCrypt – A Cryptographic Library Implemented in Java*. Tech. rep. Bern University of Applied Sciences, Engineering and Information Technology, June 2012. URL: http://e-voting.bfh.ch/students/theses-reports.

[16] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. issn: 0001-0782. doi: 10.1145/359340.359342. url: http://doi.acm.org/10.1145/359340.359342.

[17] Claus Peter Schnorr. "Efficient signature generation by smart cards". In: *Journal of Cryptology* 4 (3 1991), pp. 161–174. issn: 0933-2790. doi: 10.1007/BF00196725. url: http://dx.doi.org/10.1007/BF00196725.

[18] Adi Shamir. "How to share a secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. issn: 0001-0782. doi: 10.1145/359168.359176. url: http://doi.acm.org/10.1145/359168.359176.

A User Handbook

This chapter describes how the MobiVote Application can be used in practice.

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. At least two participants are required in order to run a voting session.

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 (see Figure A.1).

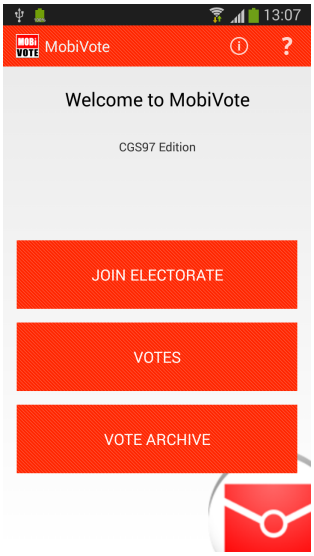

Figure A.1: MobiVote main screen

A.5 Setup a Vote

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.

Admin 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 (see Figure A.2 on the following page). In the "Vote Setup" screen (see Figure A.3 on the next page), 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.

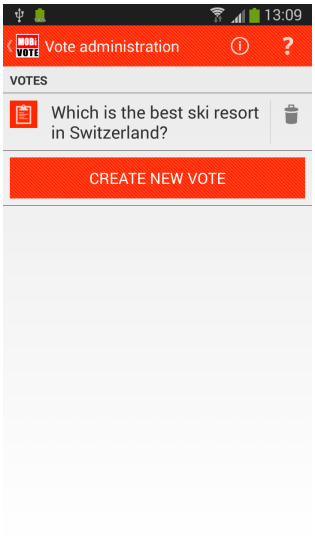

Figure A.2: Available votes

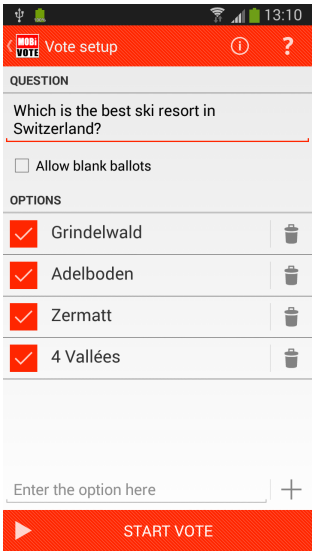

Figure A.3: Vote setup

A.6 Start a Voting Session

Admin 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 (see Figure A.4 on the following page). 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 network (see Figure A.5 on the next page). When choosing a network, you should consider that all the participants need to be allowed to connect to this network.

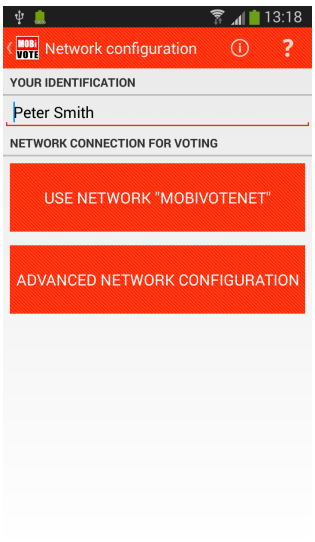

Figure A.4: Network configuration

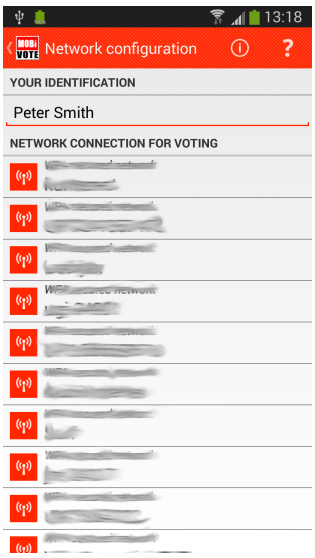

Figure A.5: Advanced configuration

A.7 Share the Session Parameters

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

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

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 Figure 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.

Figure A.6: Network information

Figure A.7: NFC tag

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 choose the option Join electorate. First, you need to enter an identification into the according textbox at the top of the screen (see Figure A.8 on page 69). 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. 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 (see Figure A.9 on the next page). 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 (see Figure A.10 on the following page). 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 (see Figure A.11 on the next page). 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.

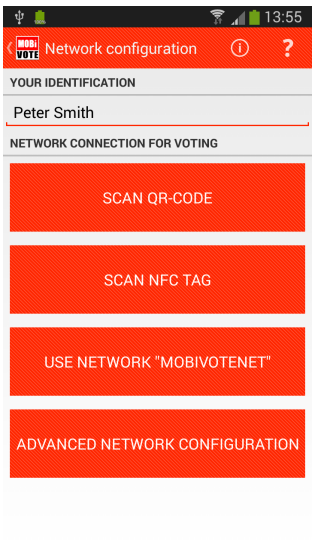

Figure A.8: Join electorate

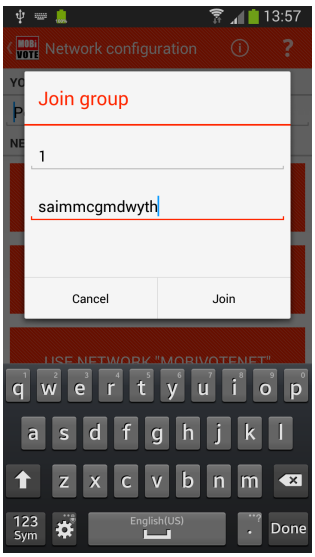

Figure A.9: Enter password

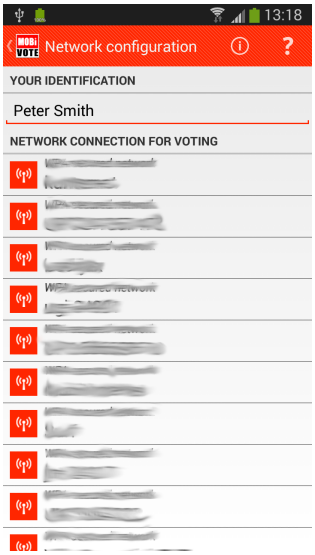

Figure A.10: Advanced configuration

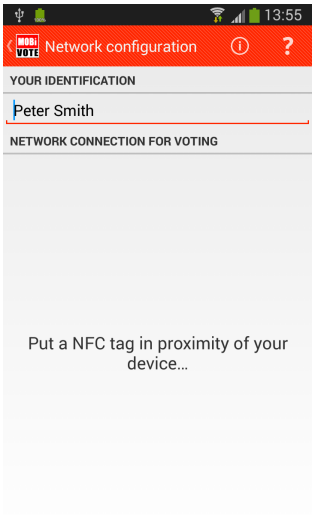

Figure A.11: Scan NFC tag

A.9 Define the Electorate

As soon as all participants have joined their devices to the voting session, the administrator can forward to the next screen by clicking on Next. In the following screen (see Figure 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 (see Figure A.13). Please note that the administrator can include or exclude himself in the electorate. Furthermore, the administrator has to define a so called *threshold* value. This value defines how many honest behaving participants need to be available when the vote is tallied. This value has to lie between two and the number of participants in the electorate. The value can be adjusted using the slider at the bottom of the screen.

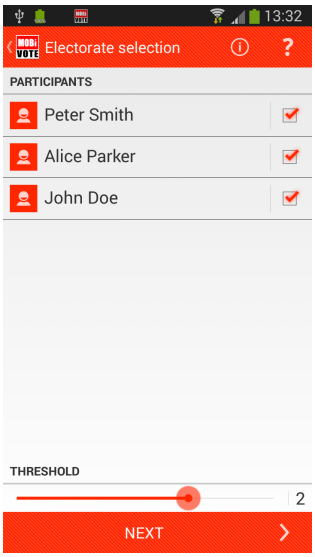

Figure A.12: Administrator defines electorate

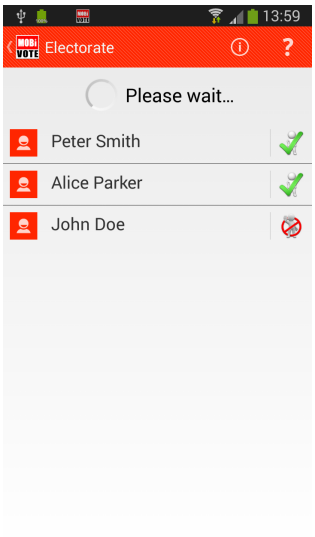

Figure A.13: Display electorate

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 (see Figure A.14 on the following page), 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.

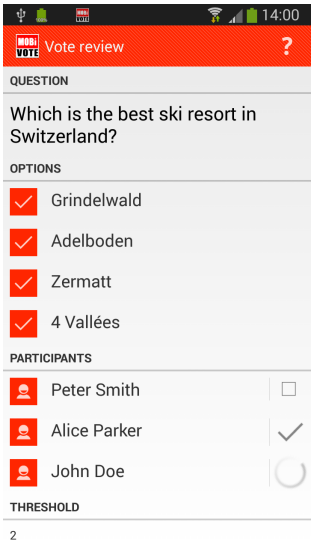

Figure A.14: Vote review

A.11 Voting

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

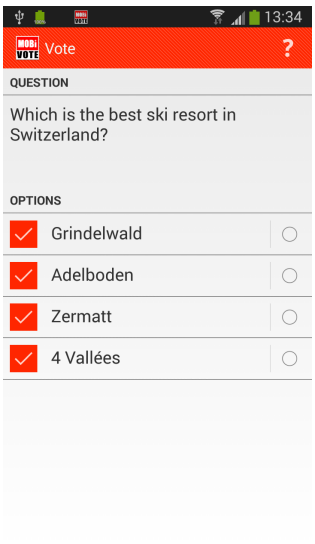

Figure A.15: Vote screen

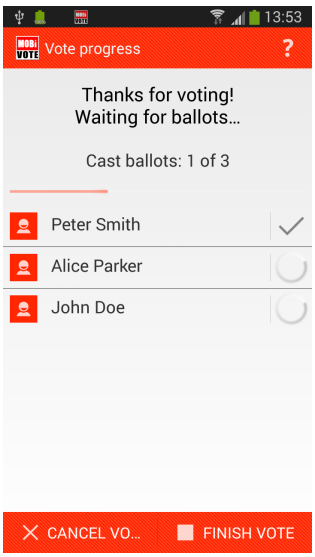

Figure A.16: Waiting for ballots

A.12 Display the Result

After a successful tally, you will be redirected to the result screen (see Figure A.17 on the next page) 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 rounds are required to reach a decision.

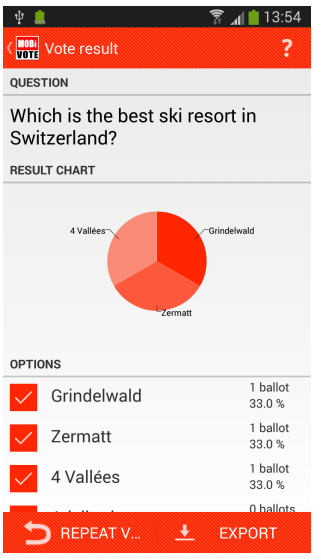

Figure A.17: Result of the vote

A.13 Vote Archive

The vote archive (see Figure A.18 on the following page), 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.

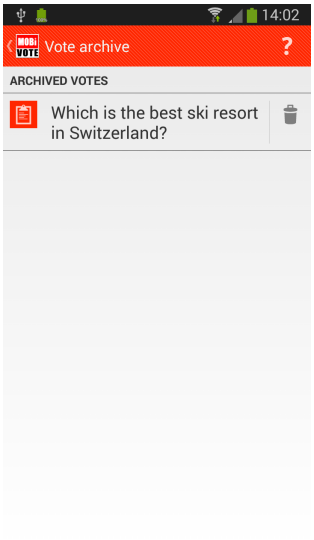

Figure A.18: Vote archive