Electronic Voting with Fully Distributed Trust and Maximized Flexibility Regarding Ballot Design

Oksana Kulyk, Stephan Neumann, Melanie Volkamer, Christian Feier, Thorben Köster
Technische Universität Darmstadt / CASED, Germany
Email: firstname.lastname@cased.de

***Abstract*—One common way to ensure the security in voting schemes is to distribute critical tasks between different entities — so called trustees. While in most election settings election authorities perform the task of trustees, elections in small groups such as board elections can be implemented in a way that all voters are also trustees. This is actually the ideal case for an election as trust is maximally distributed. A number of voting schemes have been proposed for facilitating such elections. Our focus is on a mix net based approach to maximize flexibility regarding ballot design. We proposed and implemented a corresponding voting scheme as an Android smartphone application. We believe smartphones are most likely to be used in the election settings that we consider in the paper. Our implementation also enables voters to remotely participate in the voting process. The implementation enables us to measure timings for the tallying phase for different settings in order to analyze whether the chosen mix net based scheme is suitable for the considered election settings.**

I. Introduction

Recently there has been an increased interest in remote electronic voting, with a focus on large scale elections. However, there are also many smaller scale elections, such as polls in private associations, university environments, committees, and boards with 20 to 30 voters. These boards used to conduct their elections during meetings on paper. Some are planned in advance and others are spontaneous, some use simple yes / no ballots, others more complex options including write in ballots. Elections and polls during meetings are challenging because they happen frequently and people people's mobility has increased. This means that voters are sometimes not present to vote on paper in person. So far technology enables them to participate in public discussions (e.g., over video conference), but they are then either excluded from the voting process or they have to sacrifice the secrecy of their vote in order to participate.

Remote electronic voting would enable them to participate in secret elections, even when they are not physically present. However, well known remote electronic voting schemes such as Civitas/JCJ [1] and Helios [2], [3] are not appropriate as these schemes distribute the duties of registration, voting and tabulation among a number of entities, in advance, requiring a long and time-consuming preparation phase. All this imposes a financial and administrative burden on the election authorities which seems not to be adequate for small scale board elections, in particular spontaneous board elections.

Thus, what is required is a distributed voting scheme, without central servers utilising only the voter's own devices, be it their laptops or smartphones. Note, besides not relying on central servers and not requiring lengthy preparation processes, distributed voting schemes have a further advantage: trust is distributed amongst all voters as all act as trustees.

Correspondingly, our contribution is the proposal of a voting scheme that meets all the above-mentioned requirements of secret elections and polls. The proposed voting scheme is based on existing cryptographic components used in centralized voting schemes such as verifiable mix nets, verifiable secret sharing and threshold decryption.

Furthermore, we implemented the corresponding scheme as an Android smartphone application, allowing voters to participate remotely. Note that we selected smartphone applications as smartphones are most likely to be used in the contemplated election setting and are, as such, the worst-case scenario regarding limitations with respect to computation and network capacity. The implementation enables us to measure timings for the tallying phase for different settings in order to analyze whether the chosen mix net based scheme is acceptable for the considered election settings.

The remainder of this paper is structured as follows: Section II outlines the requirements that were determined to be of relevance for the present election setting. In Section III, we present the design decisions and components selected throughout the voting scheme development process. In Section IV, we describe the composition of these components in terms of a scheme description and evaluate the scheme's security in Section V. In Section VI, we report on the implementation process. Section VII analyzes the scheme's efficiency. Section VIII reviews the related work and Section IX concludes.

II. Requirements

Based on discussions with potential boards (i.e. customers), we identified the following general and security requirements for a suitable voting scheme. Note, these requirements should be considered from a practical perspective since different (often unclear) legal requirements hold in such election settings than for national elections.

A. General requirements

The following general requirements were identified:

Ballot flexibility: It should be possible to conduct elections with ballots of any complexity due to the high spontaneity of corresponding polls:

- Yes/No election
- Multiple candidate selection ("k out of L" election)
- Priority voting (ranking of the candidates)

- Write-in ballots.

Voter flexibility: It should be possible to change the list of eligible voters for each new vote.

Spontaneity: Conducting the election should require as little preparation as possible.

Mobility: The application should run on everyday mobile devices.

Remote participation: It should be possible to cast a vote without being physically co-present with the other voters.

Usability: The system should be usable by non-experts.

Efficiency: The tallying phase should not take more than 15 minutes for 25 participating voters.

Furthermore, it cannot be assumed that it is possible to use PKI.

B. Security requirements

The following security requirements have been identified:

Eligibility: The system should only accept votes from eligible voters.

Uniqueness: Only one vote should be accepted from each voter.

Fairness: The voter should be unable to see the election results, complete or partial, before she casts her own vote.

Vote secrecy: It should be impossible link a voter to his or her individual vote.

Integrity: It should be impossible to replace a cast vote with a vote for another option.

Verifiability: The voter should be able to verify, that the vote she intended to cast is included in the final tally (*individual verifiability*). Furthermore, any third party should be able to verify, that all the cast votes have been tallied correctly (*universal verifiability*).

Robustness: After the votes have been cast, the system should be able to fulfil its functions and tally the votes despite minor errors.

These security requirements should be ensured in the following security model. It is assumed that:

1) More than the half of all the voters are honest and available during the whole voting process i.e. vote casting and tallying. This assumption is justified due to the fact that it would be unreasonable to conduct an election where the majority is corrupt.
2) The devices belonging to honest voters are also reliable and trustworthy, and are not affected maliciously by faults in hardware or software (operating system and voting application). This assumption is justified for the same reason as the previous one. Honest voters without honest devices cannot feasibly run the protocol in an honest way. Note, that in certain settings this assumption might be difficult to ensure. Namely, the smartphones are obviously used privately for other purposes, and might be at risk of infection with malware, especially when the owner is not an expert in mobile security and does not take security precautions. For example, if the OS version on the smartphone is not up-to-date, and the owner often installs apps from untrusted sources, the risks of running the election on such smartphones might be too high, and the application should not be used.
3) Honest users' devices are able to communicate with each other. Similar to the previous assumption this assumption enables honest voters to run the election.
4) No coercion takes place.

To facilitate the second assumption, it is important to embrace diversity in software and hardware. There are several manufacturers of the Android smartphones, thus, there is at least some degree of diversity. The diversity in software can be ensured, if there are different sources and a number of software developers, where the voters can download the voting application from.

Note that we can only guarantee the security requirements for honest voters. However, this holds true for traditional elections as well. For instance, a malicious voter cannot be prevented from forwarding her mail voting material to another person, thus breaking the uniqueness property.

III. Design Decisions

In this section we discuss the cryptographic primitives we used in the proposed voting system.

A. Public Key Infrastructure

As we cannot assume that PKI is in place, part of the voting application is to establish one. We do this by first exchanging the voters' RSA public keys for message authentication, and then exchanging the voters' AES keys for message encryption. One must provide protection against the man-in-the-middle attacks while exchanging the RSA public keys. One way to do this, without relying on certificate authorities and other rather complex preparations, is to use the key exchange based on short authentication strings, as described in [4]. The scheme relies on the existence of an out-of-band channel — namely, the voters should be able to communicate with each other either via physical proximity, or via video or telephone call. This channel is then used to perform manual verification of short strings over such a channel in order to frustrate man-in-the-middle attacks. In order to improve the **usability** of this verification, according to the proposition in [5], the strings have 24-bit length, and are represented as passphrases of three words from the from the PGP Word List [6]. Note, that the communication channels between eligible voters have to be established beforehand in order to execute this scheme; other preparations are not needed, thus increasing **spontaneity**.

After we use the scheme for exchanging the RSA public keys between the voters, thus providing means for message authentication, these keys are then being used to securing communications while establishing symmetric AES keys between each pair of voters via Diffie-Hellman key exchange. For generating the secret parameters in the Diffie-Hellman exchange, the SHA-256 is used as the key derivation function. Thus, means for securing end-to-end encryption are provided.

B. Verifiable secret sharing and threshold decryption

Almost all proposed electronic voting schemes rely on a distributed verifiable secret sharing scheme to generate the election key in a distributed manner and a verifiable threshold

decryption scheme to decrypt individual votes or the sum of all votes in a distributed manner.

A number of secret sharing schemes have been proposed in the literature ([7], [8], [9], [10]), while some of them do not have the means to verify the correctness of the secret sharing, or require the existence of a single trusted instance for key distribution. The scheme that does not have these disadvantages is the one described by Pedersen in [11], [12] and is proven to be IND-CPA secure if used in conjunction with the ElGamal cryptosystem, as shown in [13]. Thus, we decided to use this approach in our application. The corresponding verifiable threshold decryption scheme, which relies on the keys being generated as in [11] is described in [14].

C. Homomorphic tallying versus mix net approach

The approaches most commonly used in electronic voting schemes for preserving the vote secrecy are the homomorphic tallying (e.g. in [15], [14]) and mix net schemes (e.g. in [16], [17]). The first approach relies on homomorphic properties of a crypto system used to encrypt the votes, most commonly, the exponential ElGamal. The homomorphic property is used to multiply the encrypted votes, and then to decrypt the resulting sum. This approach is inefficient for complex kinds of ballots such as priority ranking, and is unsuitable for write-in ballots. Therefore, for ensuring **ballot flexibility** in our application we chose to use the mix net approach.

Two types of mix nets have been proposed: decryption mixnets (e.g. in [18], [16]) and re-encryption mix nets (e.g. in [17], [19]). In order to ensure **robustness** of the scheme, we decided to implement one of the re-encryption mix net schemes. Note, in case of a decryption mix net, one dishonest note can violate robustness.

These schemes also rely on the homomorphic property of an underlying crypto system. A number of entities called the *mix nodes*, the role of which is taken by the voters in our setting, participate in the scheme, whereby each mix node in turn shuffles the list of encrypted ciphertexts $C = (c_1 = Enc_h(v_1, s_1), ..., c_N = Enc_h(v_1, s_1))$ using a secret permutation π and secret randomness values $r = (r_1, ..., r_N)$, outputting the shuffled list $C' = (c'_1, ..., c'_N)$ so that holds:

$$c'_i = Enc_{pk}(1, r_i) \cdot c_{\pi(i)}$$

D. Verifiable mix net schemes

In order to ensure **integrity** and to provide **verifiability**, however, each note has to prove that the input and output set contain the same votes (without revealing π and r). A number of schemes for providing a so called non-interactive zero-knowledge proof of shuffle have been developed ([20], [21], [22], [23], [24]) which mainly differ in their efficiency, degree of vote secrecy, integrity/verifiability as well as robustness. In order to decide which of the proposed proofs is the most appropriate one for our setting, we compare them wrt. efficiency, vote secrecy and integrity/verifiability. For the comparison we apply the following considerations:

- For the efficiency considerations, we consider the number of modular exponentiations E needed for computing the proof of shuffle and for verifying it.
- In order to measure the degree of secrecy of the proposed mix net schemes, we consider the size of *anonymity group* $|A|$. Let $C = \{c_1, ..., c_N\}$ be the list of ciphertexts that results from the final shuffle. Let $A \subseteq C$ be a group of ciphertexts, whereby it is known that the vote of some given voter is in A. Ideally, this group would be the group of all votes cast within the election ($|A| = N$), in which case it is said that a mix net provides *complete* secrecy. Otherwise, if $|A| < N$, the mix net's secrecy is *incomplete*.
- In order to measure the degree of integrity/verifiability of a mix net scheme, we consider the probability p, that the attacker can successfully prove the correctness of an incorrect shuffle. Note, in case p is negligible, the mix net scheme provides *overwhelming* integrity.
- In order to measure the degree of robustness, we consider
  the minimal number of voters t, that should participate and behave correctly during the mixing, in order for it to provide a valid result.

The result of the evaluation according to these considerations is proposed in Table I. As one can see, the schemes that provide the best efficiency, such as the schemes in [20], [21], are seriously lacking in either secrecy or integrity, in particular, for small values of N. As such, the proof of shuffle with the best trade-off between security (secrecy, integrity/verifiability, robustness) and efficiency is the one proposed in [23]; however, since it is covered by patent - to the best of our knowledge, we chose to use the method proposed by Wikström in [24], [25] in our implementation.

TABLE I: Comparison of mix net schemes

PoS	$\|A\|$	E	p	t
[20]	$N/2$	$2N$	50%	$(N/2+1)$
[21]	$complete$	$6\sqrt{N}$	$(\sqrt{N}-1)/N$	1
[22]	$complete$	$12N$	$overwhelming$	1
[23]	$complete$	$2N \log k + 4N$	$overwhelming$	1
[24], [25]	$complete$	$20N + 19$	$overwhelming$	1

k is a divisor of N

E. Proof of Correctness

As shown in [26], ensuring vote secrecy also depends on whether ballot independence is assured: namely, a malicious voter should be unable to cast a vote which is both valid and meaningfully related to a cast vote of another voter. In particular, a group of malicious voters of size f can attempt to break vote secrecy by taking a vote cast by another voter, and casting it as their own vote. Then, after looking at a final result, they could see which vote has been cast at least $f + 1$ times, thus figuring out how the attacked voter has voted. A simple way to prevent this attack is to make the voters prove that they know a corresponding plaintext for a ciphertext message they cast as their vote. For the ElGamal encryption, this can be done by using the non-interactive proof of knowledge of discrete logarithm (described in [27]). Thus, for $c = (a, b) = (g^r, v \cdot h^r)$

with g, h being the ElGamal public keys, the voter has to prove the knowledge of r given a.

IV. Voting Scheme Description

The voting scheme consists of following basic components: verifiable secret sharing, re-encryption mix net, and verifiable distributed decryption. As a crypto system used in encrypting the votes, we chose ElGamal due to its homomorphic properties and its wide use in the selected schemes. Let p, q, g be the corresponding ElGamal parameters, that are publicly available.

1) Ballot initialization: The initiator of the voting composes a ballot that, according to the election type, may consist of the voting question, possible answers, voting rules etc. The empty ballot is then broadcast to all the voters chosen by the initiator, whereby each voter has an option either to agree to participate in the voting, or decline. As a result, the group of voters that is about to participate in this election is formed. In case a set of keys for the election (see Section III-B) has already been generated for this group, the voting proceeds with the *vote casting* stage; otherwise, it proceeds with the *key exchange* stage.

2) Key exchange: This phase consists of generating keys for the election via a verifiable decentralized threshold secret sharing scheme described in [11] with threshold value of $\lfloor N/2 \rfloor + 1$: x_i, the shares of private key that each voter holds, and the jointly computed public key h. The participants also exchange commitments h_i to x_i, which are calculated as $h_i = g^{x_i}$, that are later used for verifiable decryption. The key exchange phase only needs to be performed once for each group of voters; in any further elections conducted by the same group, the previously generated keys can be securely reused.

3) Vote casting: The voters are given a certain time limit, during which they are supposed to cast their vote. The vote v_i is encoded so that it could be used as a plaintext in ElGamal encryption, and e_i is calculated as $Enc_h(v_i, r_i)$ for a random $r_i \in_R \mathbb{Z}_q$. Furthermore, the proof of correctness is used to demonstrate the knowledge of v_i to prevent ballot-copying attacks, as shown in III-E. After (c_i, p_i) have been broadcast by all voters, each voter possesses the initial list of all votes $C_0 = (c_1, ..., c_N)$.

4) Tallying: At the beginning of the tallying phase, the votes are anonymized (Figure 1): this process is divided into N rounds, with fixed execution times. In each round, the voter i applies a mix net scheme to the list C_{i-1} using a random vector $r = (r_1, ..., r_N)$ and a permutation π in order to get a shuffled list $C_i = Enc_h(1, r) \cdot (C_{i-1})_\pi$. She also computes a non-interactive proof of shuffle P_i as described in Section III-D, in order to demonstrate that the shuffle has been executed correctly. After that she communicates the values (C_i, p'_i) to other voters. Then, each one of the remaining voters verifies p'_i, and if it is verified, accepts C_i; if p'_i is not verified, or if the voter i does not send any shuffle result within a round time, sets $C_i := C_{i-1}$. At the end, after all the voters have performed the shuffling, the list C_N is accepted as the final list of anonymized votes. The verifiable decryption scheme is then being executed as described in [14] (Figure 2): for each encrypted vote $c_i \in C_N, c_i = (a_i, b_i)$ each voter j computes the partial decryption share $d_{i,j} = a_i^{x_j}$ using her private key share x_j. (S)he then also computes the non-interactive zero-knowledge proof $p''_{i,j}$ to prove that the secret value x_i used for partial decryption is the same value, that was committed to during key exchange phase. The voters then broadcast their computed values (d_j, p''_j) with $d_j = (d_{1,j}, ..., d_{N,j})$, $p''_j = (p_{1,j}, ..., p_{N,j})$. As soon as any voter gets a threshold amount of partial decryptions and proofs of its correctness $(d_{i,j}, p''_{i,j})$, whereby $p''_{i,j}$ is verified successfully, she can reconstruct the decryption of c_i from the collected values of partial decryption shares. In this way, all the votes in C_N are being decrypted, resulting in values of $V = (v_1, ..., v_N)$. The final result is then tallied according to election rules: as such, for example, if each vote represents a candidate from the given list $v_i \in \{C_1, ..., C_L\}$, the result is the sum of the votes cast for each candidate, $S = (s_1, ..., s_L),\ s_i = |v_j : j = 1, ..., N, v_j = C_i|$.

V. Security Analysis

This section is dedicated to an informal security argument on the presented scheme. To evaluate its security, we identify threats against the security requirements (see Section II-B) and show that the scheme defends against these threats under given assumptions. Note, that the scheme can only provide defence against these threats for the voters with uncorrupted devices, as otherwise the application would just behave according to the attacker's commands, instead of following the scheme.

Eligibility A non-eligible voter can cast the vote in the system, in case there is no authentication in place, or the voter can fake her identity and impersonate an eligible voter. This is not the case if the list of all voters is known in advance, which is ensured in the ballot initiation stage, and if reliable PKI exists, providing means for message authentication and thus preventing identity impersonation. Therefore, it should be impossible for the attacker to impersonate an eligible voter and cast a vote instead of her.

Uniqueness In case no votes from non-eligible voters are accepted, which is ensured via eligibility, a voter can break uniqueness and cast more than one vote, if she can fake her identity and pretend to be another eligible voter. This is impossible due to existing PKI. Thus, it can be ensured that during the vote casting stage, only the voter's first vote (alternatively, only the last one) is accepted.

Fairness In the scheme the fairness property can be broken if a voter is able to reveal others' votes during vote casting. To do this, s/he must be able to decrypt the votes that are broadcast. This is only possible, if at least $\lfloor N/2 \rfloor + 1$ voters collaborate and use their secret keys for decryption. This is impossible according to the assumptions 1-2 in Section II-B; therefore, there is no way for any voter to know the intermediate result at vote casting.

Vote Secrecy The possible ways to break secrecy in the scheme is to either decrypt the cast votes before they are anonymized, or to prevent them from being anonymized. The first way is possible if at least $\lfloor N/2 \rfloor + 1$ voters cooperate maliciously and use their secret key shares for

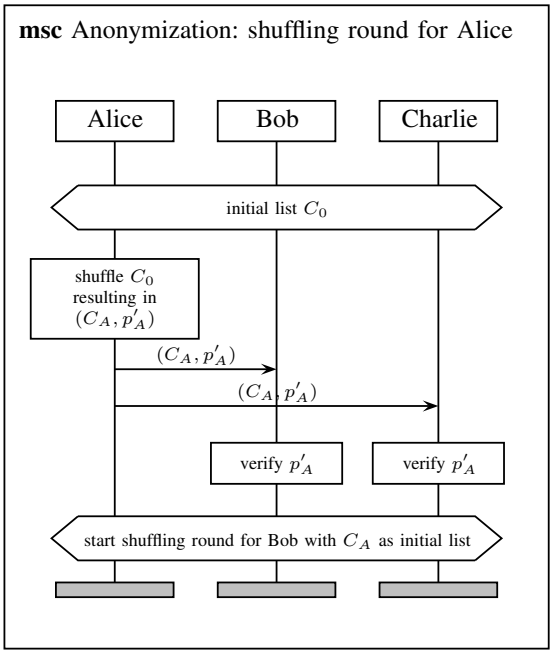

Fig. 1: Anonymization round for $N = 3$

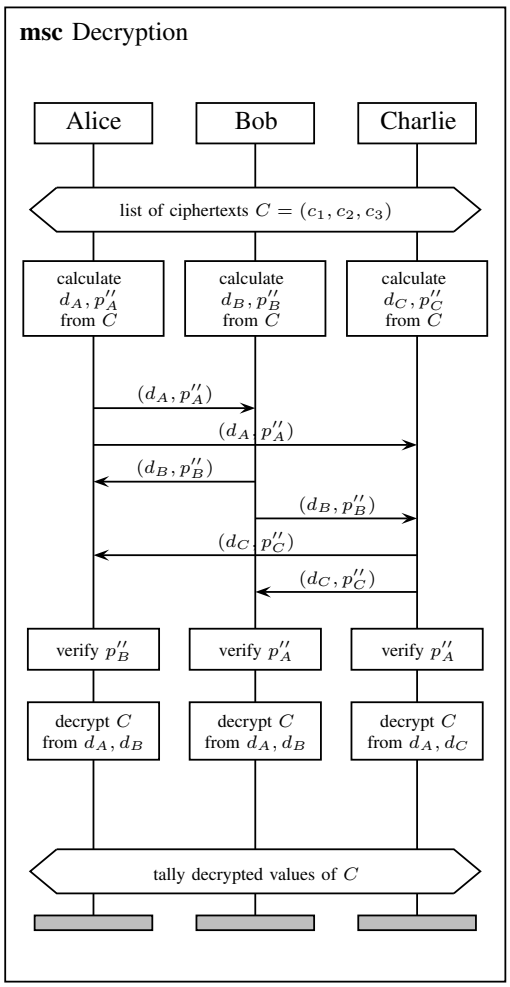

Fig. 2: Decryption for $N = 3$

decryption. The second way is possible if all but one[1] voter decline to perform the anonymization or to keep the correspondences between input and shuffled ciphertexts a secret. Thus, according to assumptions 1-2 in Section II-B, vote secrecy is ensured.

Integrity A way to break integrity and replace some cast vote with another vote, would be either to replace the ciphertext during anonymization stage, or to provide a manipulated partial decryption during tallying stage. This attempts will be detected, however, due to the employment of zero-knowledge proofs during decryption and anonymization, which each voter has to verify before accepting. Therefore, everyone should have the possibility to verify the correctness of the tallying. Thus, any manipulation with the election result will be noticed.

Verifiability Similarly to ensuring integrity, universal verifiability of the correctness of election result is ensured due to non-interactive zero-knowledge proofs that could be verified by anyone using publicly available information. Given universal verifiability, the only way to break individual verifiability would be for the application to cast a vote that is different from the voter's intention. However, due to the assumption 2 in Section II-B, individual verifiability is ensured.

Robustness The result of the voting cannot be decrypted and thus tallied, if only less then $\lfloor N/2 \rfloor + 1$ voters are available and can communicate with each other during decryption. Additionally, the result cannot be tallied without necessarily breaking vote secrecy, if the anonymization of the votes has not been performed correctly, which is possible, as described above, if all but one voter are unable to shuffle the ciphertexts and keep the correspondences between the input list and the shuffled list secret. Therefore, according to assumptions 1-3 in Section II-B, robustness of the system is ensured.

[1] If only one voter is honest, then the public will not know the correspondences between the voter's identity and the vote; however, if all the other voters are dishonest, and each dishonest voter i reveals the correspondences between the ciphertexts in lists C_{i-1} and C_i to the public, the honest voter will be the one who knows how each one has voted. Thus, vote secrecy during anonymization could be ensured only if at least two voters perform their shuffling correctly and do not reveal the correspondences between the ciphertexts.

VI. Implementation

In this section we describe the implementation details of the voting scheme, as well explain particular design decisions

we made.

A. Design Decisions

1) Android app: We developed an application to implement the described voting scheme for Android smartphones. Android is based on a Linux Kernel and is the most widely used mobile operating system. It runs on many different machines which differ in many respects like, for example, screen resolution, CPU power and available memory. The application is designed to support all machines which run Android 4.0 or higher and have more than 512 RAM available.

2) Communication: To establish the communication channels between the voters' smartphones, we had to choose between several options, such as BlueTooth, WiFi-Direct, SMS or instant messaging protocols such as MSN or ISQ. We chose to use XMPP, which is an open-source instant messaging protocol. In advantage to other options, it allows for communications over the network without being in physical proximity to each other, does not place substantial restrictions on message length, and can be extended thus making it easier to adjust for our implementation. To establish a connection to other participating smartphones the Smack API[2] which builds upon XMPP is used. In order for the voters to communicate with each other, the XMPP server has to be available, either as a public server, or as a private server, established by the company. The voters then use their account data on this server to log in the application. As the XMPP protocol communicates via network, **remote participation** is ensured, by enabling every eligible voter to participate in the voting, as long as she has access to network connection, for example, to the mobile internet on her smartphone.

For establishing the PKI we prepared a central server that is used as a "bulletin board" where the initial list of voters is stored. The bulletin board is needed for establishing the PKI only, and is not required on any other stage of voting. This initial list of voters is required in order to enable the initial communication between voter's devices, as voter could send the messages to others only knowing their XMPP account IDs.

This server is relied on with regards to availability only, and does not hold any sensitive information. We use the scheme described in III-A in order to exchange the RSA keys and the AES keys between the voters.

3) Libraries: For implementing the mix net, we did not use the Verificatum implementation by Wikström[3] due to licence restrictions. Instead, the application uses the open-source `unicrypt`[4] library for the mix net implementation. We used the `guava-library`[5] as a utility library e.g. for Base64 encoding. Android ships with a cut-down bouncycastle implementation for cryptographic primitives which only allows symmetric encryptions up to 128 Bit. To support better encryption schemes like 256 Bit symmetric encryption an external library called `spongycastle`[6] is used. Spongycastle is a derivation of Bouncycastle[7], the most popular and extensive Java library for cryptography, which is optimized for Android and renames the packages to avoid classloader conflicts.

B. Walkthrough

We have attempted to make the user interface as simple as possible, requiring only the minimum amount of interaction from the users. We also iteratively improved them due to feedback from colleagues and friends. Note, we plan as future work to evaluate the usability within a user studies.

When starting the application, the voter arrives at the welcome page and logs herself in using her XMPP account. After logging in, the user is referred to the Main Menu (see Figure 3). There, the PKI establishment process can be launched, which concludes when all the voters comparing and verify the passphrases displayed on their screens (see Figure 4). Note, that the PKI establishment scheme is only performed once for each set of voters. It is only repeated when new persons (i.e. new employers, or new boardroom members) are added to the list of eligible voters.

After the PKI has been established, the elections can be conducted. The person who wants to start the election composes and broadcasts the ballot as seen in Figure 5. As all other participants see the invitation and agree to participate, the election starts: if this group of voters starts an election for the first time, the key exchange is being run first. Otherwise, the voters can start with the vote casting, whereby each voter selects her vote and confirms the vote as seen in Figure 6.

After all votes are received the mix net starts anonymizing the votes. As this is the most computationally intensive part of the process, it may take some time. Afterwards the votes are decrypted and tallied and the result shown as seen in Figure 7.

A flow diagram which explains the PKI establishment process (Figure 8), ballot initiation (Figure 9), and voting process (Figure 10) are given, while the captions in bold on the diagrams refer to the steps where the interaction of the voter with the user interface is needed.

C. Fault Handling

We have identified the steps of the voting process, whereby some faults might be present. Most commonly some voters not being present or being unable to communicate with the others might occur. We have already shown, in Section IV, how some of these faults are handled. Furthermore, as shown in Section V, some of these faults, such as the voters failing to produce valid partial decryptions of a vote, could be ignored under the assumptions that we make.

Other faults are the ones that occur during voting phases, that preclude the tallying stage: namely, faults could occur during PKI establishment (i.e. the adversary trying to execute a man-in-the-middle attack), ballot initialization stage (such as voters not responding to the invitation to vote), or vote casting. The diagrams in figures 8,9,10 show the way the application is supposed to handle these faults. As such, for example, the voter who wishes to initiate the election has the option to decide, whether she still wants to start the election if not all of the invited voters respond to her invitation, or to wait some more for the missing voters to respond, or to cancel the election.

[2]http://www.igniterealtime.org/projects/smack/
[3]http://www.verificatum.org/
[4]https://github.com/bfh-evg/unicrypt/
[5]https://code.google.com/p/guava-libraries/
[6]http://rtyley.github.io/spongycastle/
[7]https://www.bouncycastle.org/

Fig. 3: Main Menu

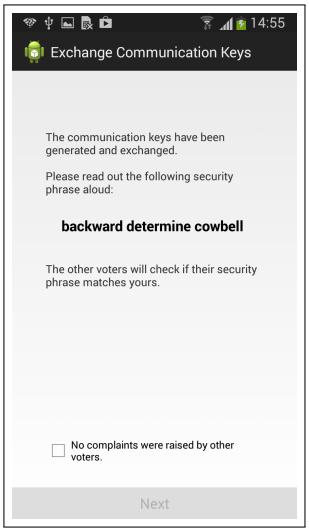

Fig. 4: Establishment of the PKI

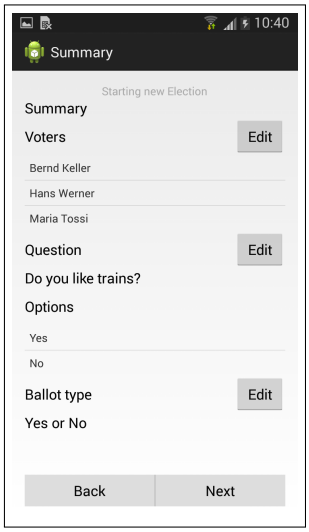

Fig. 5: Summary of the ballot for the new election

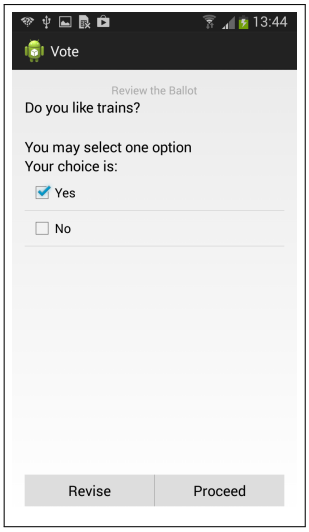

Fig. 6: Overview of a cast vote

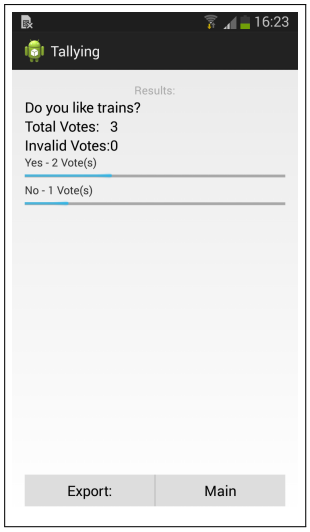

Fig. 7: Election result

Another source of faults during the voting, is the inconsistency of message broadcast. In order to broadcast a message using XMPP, the message has to be sent separately to each receiver. Thus, it makes the system vulnerable to Byzantine faults, whereby a malicious voter can send different messages to different receivers (for example, during broadcasting a cast vote), thus endangering robustness of the voting. One way to solve this problem is to make the voters manually compare the result of each stage (for example, by comparing hash values of a complete list of cast votes at the end of vote casting). Another solution is to implement additional communication schemes that ensure Byzantine Fault Tolerance, such as the schemes described in [28], [29][8].

[8]Note, that some of the methods to implement BFT provide more efficiency at the cost of requiring additional assumptions regarding the amount of faulty nodes f out of total N, most commonly, $f < \lfloor N/3 \rfloor$.

VII. Efficiency Evaluation

Without counting the costs of the communication (i.e. signing and verifying the communicated messages, as well as encrypting/decrypting them when needed), the cost of the execution of the scheme in number of required modular exponentiations, with the anonymization stage being the most computationally extensive part, is as follows:

$$26N^2 + 22N + \lfloor N/2 \rfloor + 1 + N(\lfloor N/2 \rfloor + 1) - 1$$

Thus, the efficiency of the voting scheme is $\mathcal{O}(N^2)$. Note that it only depends on the number of the voters, and not on ballot complexity, such as number of candidates or possible options.

As additional computational and communication costs arise in the implementation, which depend on programming tech-

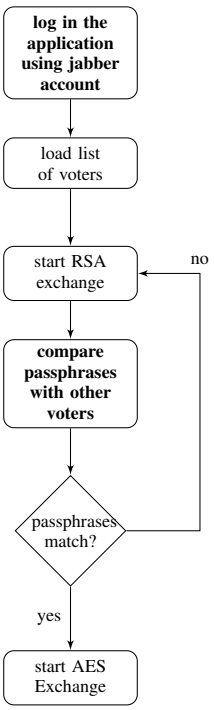

Fig. 8: Establishment of the PKI

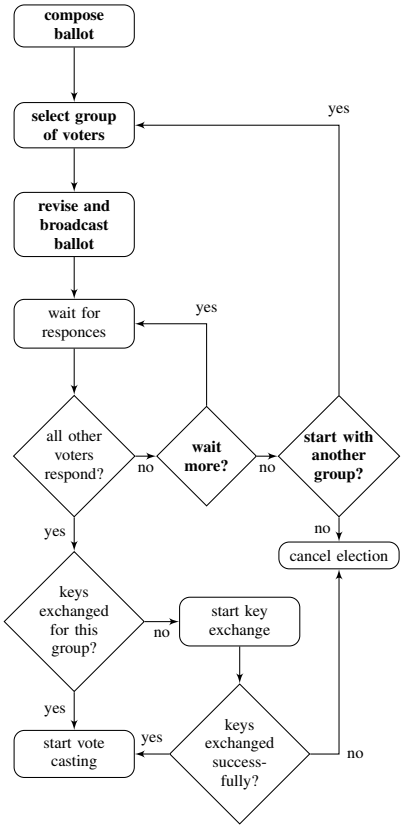

Fig. 9: Ballot Initiation

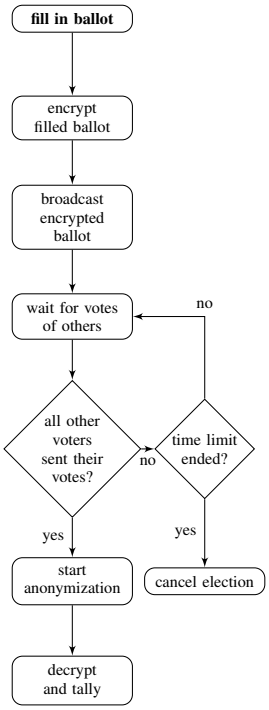

Fig. 10: Vote Casting

niques and network capabilities, we evaluated the performance of the application, by measuring the time it takes to calculate and display the result of voting after the votes have been cast. The application was run on several S3 Samsung smartphones, all in the same room. The "voters" were represented by Gmail accounts with GTalk as the XMPP server for communication, created for test purposes. We did not count the time taken for the PKI establishment stage, since it is only conducted once initially, nor the key exchange stage, since it only has to be executed once for a group of voters. We also did not record the time elapsed during ballot initialization and vote casting, since the time spent on this stage depends mostly on how long the voters take to make their decisions and cast their votes. The key length is as follows: the RSA keys used for message authentication have 2048-bit length, as well as the ElGamal parameters g, p. The ElGamal secret keys, as well as random values used in exponentiations, have 256-bit length.

The resulting times from running the election between $2-5$ voters are given in table II. The times seem linear because of how the cryptographic schemes with several rounds have been implemented in order to achieve synchronization: each round is given a fixed amount of time, during which it is expected for all computations to be complete. Thus, this time is chosen as an upper limit for the computations - namely, for the mix net scheme, the duration of one shuffling round is set such as one should be able to complete the shuffling of 25 ciphertexts, which includes calculating and verifying the corresponding proofs of shuffle. Thus, the time spent on anonymizing the votes is $\mathcal{O}(N)$ for $N \leq 25$. The time for decrypting the votes is $\mathcal{O}(N^2)$, but it is relatively small compared to the anonymizing stage. Thus, extrapolating the times for 25 voters[9], we can assume that the election will last slightly less then 12 minutes on such devices.

TABLE II: Execution times of tallying stage

Number of voters	Average execution time (ms)	Average execution time (min)
2	65764.5	1.10
3	85152.7	1.42
4	109375	1.82
5	129702.6	2.16

VIII. Related Work

A number of schemes for decentralized voting with distributed trust has been proposed in the literature. Among them are the works in [15], [30] and [14], which were implemented in the *MobiVote* application. The security model of these schemes is similar to the one that we describe in this paper, namely, the security of the scheme depends on the majority of the voters and their voter devices being uncorrupted. However, the schemes in question employ homomorphic tallying, thus being less suitable for complex ballots. An Android application for spontaneous decentralized voting in classroom setting has been proposed in [31]; the approach, however, does not ensure verifiability. A scheme for decentralized voting

[9]We used the polynomial trend line function in Excel.

has been described in [16], and then expanded in [32]. The scheme uses mix net scheme for anonymizing the votes; however, it relies on all the voters being uncorrupted during the anonymization stage for ensuring robustness and integrity, which is a disadvantage compared to our approach.

IX. Conclusion and Future Work

We have presented a scheme for decentralized voting with distributed trust, and an application that implements this scheme, thus enabling secure elections in small groups. We have shown that this application fulfills the security requirements of eligibility, uniqueness, fairness, vote secrecy, integrity, verifiability, robustness, as well as the general requirements of ballot flexibility, voter flexibility, spontaneity, mobility, remote participation that we have set as our goal. As a future task, we will work on the usability of the application, conducting user studies and improving the user interfaces. As part of improving usability, we will work on further improving efficiency of the application. This includes (1) using the fact, that the mix net scheme developed in [24] is specifically designed with an "offline" and "online" phase, whereby the offline phase is the computationally extensive one, and can be executed before the election actually starts. Currently, these two phases are executed one directly after another during vote anonymization. The offline phase, however, could be completed in advance, during the idle time of the protocol, when no other extensive computations are being executed, thus making the tallying phase substantially faster. Furthermore, (2) efficiency of the vote anonymization will be further improved by only requiring a subset of all voters to participate as mix nodes. We have shown that at least two honest voters are needed to ensure vote secrecy during vote anonymization. Thus the set of shufflers must contain at least two honest voters. According to our assumptions, at most $\lceil N/2 \rceil - 1$ voters are dishonest. Adding two honest honest voter upon $\lceil N/2 \rceil - 1$ results in the fact that the minimal number of voters that need to act as mix nodes is $\lceil N/2 \rceil + 1$. In order to determine the shufflers for each election, a common reference string to generate randomness can be used. One could instantiate the common reference string by a cryptographic hash value of all the votes cast in the election, then using it as an input in a deterministic function that outputs a sequence of shufflers. Another way would be to sort the list of all voters in the election according to canonical order, and choose the first $\lceil N/2 \rceil + 1$ from the sorted list. Another way to improve efficiency will be to use elliptical curves instead of integer groups, in which case additional considerations on how to encode votes are necessary.

Another direction of future work is to discuss the issue of people using same or similar smartphones as well as people all installing the software from the same vendor or download it from the same platform.

Finally, we will also have a closer look to the robustness of the application. In particular, we will implement the Byzantine Fault Tolerance scheme in order to make communication more reliable. An efficient way to do this, that requires more than two thirds of honest nodes, is described in [28]. Another, way to implement the Byzantine Agreement is described in [29]. Although this way is less efficient, it does not require changes in security model, and can be applied if more than half of all the voters are honest, provided that the means of message authentication are in place.

Acknowledgment

This paper has been developed within the project 'BoRoVo' Board Room Voting - which is funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS12054 and within the project ComVote, which is funded by the Center for Advanced Security Research Darmstadt (CASED), Germany. The authors assume responsibility for the content. We also thank the reviewers for their valuable comments that helped to considerably improve the quality of this work.

References

[1] Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: IEEE Symposium on Security and Privacy, IEEE Computer Society 354–368

[2] Adida, B.: Helios: Web-based open-audit voting. In van Oorschot, P.C., ed.: USENIX Security Symposium, USENIX Association 335–348

[3] Karayumak, F., Olembo, M.M., Kauer, M., Volkamer, M.: Usability analysis of helios-an open source verifiable remote electronic voting system. In: Proceedings of the 2011 USENIX Electronic Voting Technology Workshop/Workshop on Trustworthy Elections. USENIX. (2011)

[4] Nguyen, L.H., Roscoe, A.: Efficient group authentication protocol based on human interaction. In: Proceedings of Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis. (2006) 9–31

[5] Farb, M., Burman, M., Chandok, G., McCune, J., Perrig, A.: Safeslinger: An easy-to-use and secure approach for human trust establishment. Technical report, Technical Report CMU-CyLab-11-021, Carnegie Mellon University (2011)

[6] Zimmermann, P.R.: Pgpfone: Pretty good privacy phone owner's manual. MIT, http://web. mit. edu/network/pgpfone/manual (1995)

[7] Shamir, A.: How to share a secret. Communications of the ACM **22**(11) (1979) 612–613

[8] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: Foundations of Computer Science, 1987., 28th Annual Symposium on, IEEE (1987) 427–438

[9] Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: Foundations of Computer Science, 1985., 26th Annual Symposium on, IEEE (1985) 383–395

[10] Benaloh, J.C.: Secret sharing homomorphisms: Keeping shares of a secret secret. In: Advances in CryptologyCRYPTO86, Springer (1987) 251–260

[11] Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Advances in CryptologyEUROCRYPT91, Springer (1991) 522–526

[12] Pedersen, T.P.: Distributed provers and verifiable secret sharing based on the discrete logarithm problem. DAIMI Report Series **21**(388) (1992)

[13] Cortier, V., Galindo, D., Glondu, S., Izabachene, M.: A generic construction for voting correctness at minimum cost-application to helios. IACR Cryptology ePrint Archive **2013** (2013) 177

[14] Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. European transactions on Telecommunications **8**(5) (1997) 481–490

[15] Khader, D., Smyth, B., Ryan, P.Y., Hao, F.: A fair and robust voting system by broadcast. In: EVOTE'12: 5th International Conference on Electronic Voting. (2012)

[16] DeMillo, R.A., Lynch, N.A., Merritt, M.J.: Cryptographic protocols. In: Proceedings of the fourteenth annual ACM symposium on Theory of computing, ACM (1982) 383–400

[17] Benaloh, J.: Simple verifiable elections. In: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop, USENIX Association (2006) 5–5

[18] Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **24**(2) (1981) 84–90

[19] Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: USENIX security symposium, San Francisco, USA (2002) 339–353

[20] Jakobsson, M., Juels, A., Rivest, R.: Mix nets robust for electronic voting by randomized partial checking. USENIX security symposium (2002)

[21] Demirel, D., Jonker, H., , Volkamer, M.: Random block verification: Improving the norwegian electoral mix-net. In Manuel J. Kripp, M.V., Grimm, R., eds.: 5th International Conference on Electronic Voting 2012 (EVOTE2012). Volume 205 of LNI - Series of the Gesellschaft für Informatik (GI)., Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC, Gesellschaft für Informatik (July 2012) 65–78

[22] Groth, J.: A verifiable secret shuffle of homomorphic encryptions. Journal of Cryptology **23**(4) (May 2010) 546579

[23] Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Advances in Cryptology EUROCRYPT. (2012)

[24] Terelius, B., Wikström, D.: Proofs of restricted shuffles. In: Progress in Cryptology–AFRICACRYPT 2010. Springer (2010) 100–113

[25] Wikström, D.: A commitment-consistent proof of a shuffle. In: Information Security and Privacy, Springer (2009) 407–421

[26] Smyth, B., Bernhard, D.: Ballot secrecy and ballot independence coincide. In: Computer Security–ESORICS 2013. Springer (2013) 463–480

[27] Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3) (1991) 161–174

[28] Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. Volume 99. (1999) 173–186

[29] Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3) (1982) 382–401

[30] Hao, F., Ryan, P.Y., Zieliński, P.: Anonymous voting by two-round public discussion. IET Information Security **4**(2) (2010) 62–67

[31] Esponda, M.: Electronic voting on-the-fly with mobile devices. ACM SIGCSE Bulletin **40**(3) (2008) 93–97

[32] Alkassar, A., Krimmer, R., Volkamer, M.: Online-wahlen für gremien. DuD Datenschutz und Datensicherheit **8**(29) (2005)