Pretty Understandable Democracy 2.0 (Draft Version)

Stephan Neumann, Christian Feier, Perihan Sahin, and Sebastian Fach

Security, Usability, and Society
Technische Universität Darmstadt
Hochschulstraße 10
64289 Darmstadt, Germany
stephan.neumann@cased.de
feier@rbg.informatik.tu-darmstadt.de
perihansahin87@hotmail.com
info@sebastian-fach.de

August 16, 2014

Abstract

Technology is advancing in almost all aspects of our everyday life. One interesting aspect is the possibility to conduct elections over the Internet. However, many proposed Internet voting schemes and systems build on unrealistic assumptions about the trustworthiness of the voting environment and other voter-side assumptions. Code voting – first introduced by Chaum [Cha01] – is one approach that minimizes the voter-side assumptions. The voting scheme Pretty Understandable Democracy [BNOV13] builds on the idea of code voting while it ensures on the server-side an arguably practical security model based on a strict separation of duty, i.e. all security requirements are ensured if any two components do not collaborate in order to violate the corresponding requirement. As code voting and strict separation of duty realizations come along with some challenges (e.g. pre-auditing phase, usability issues, clear APIs), the goal of our research was to implement Pretty Understandable Democracy and run a trial election. This paper reports on necessary refinements of the original scheme, the implementation, and a trial election among the different development teams.

1 Introduction

The advance of technology, more and more, impacts our everyday life. Shopping, banking, or chatting with friends no longer depends on physical presence but may be easily done independent of time and location by digital means. In recent years, even fundamental processes of democracy have come into the focus of technological advance. Amongst the most attractive options is the possibility to conduct elections over the Internet. Since the seminal work by Chaum [Cha81], many works addressed the challenge of voting over the Internet addressing a broad set of security requirements, see for instance [LSBV10]. It turns out, however, that most of the present schemes rely on unrealistic assumptions to ensure security: for instance, the JCJ [JCJ05] scheme relies on the voter's platform being trustworthy and the Helios voting system [Adi08] relies on the voter conducting a complex verification procedure several times. The number of infected computers[1] shows that it is not realistic to rely on voters to ensure that their platforms are trustworthy. It has also been shown (e.g. in [KOKV11]) that in particular with the Helios voting system, verifiability

[1] According to [Pan14], in 2013 31.53% of all computers were infected by malware

is not accessible to voters. Furthermore, Olembo et. al [OBV13] have shown that voters do not even see the need to verify their vote due to their trust mental models.
Code voting – first introduced by Chaum [Cha01] – is one approach that minimizes the voter-side assumptions. Since its invention several code voting schemes with different advantages and disadvantages have been proposed [HS07, JRF09, RT09]. Recently, Budurushi et al. [BNOV13] proposed a new code voting based Internet voting scheme, Pretty Understandable Democracy (PUD). It ensures on the server-side an arguably practical security model based on a strict separation of duty, i.e. all security requirements are ensured if any two components do not collaborate in order to violate a corresponding requirement. Furthermore, the authors' goal was to keep the scheme as simple as possible. To date, PUD has not been implemented and therefore has only been considered from a purely theoretical perspective.
Contribution. As code voting and strict separation of duty realizations come along with some challenges for the implementation process, the election preparation and the vote casting (e.g. preauditing phase, usability issues, clear APIs), the goal of our research was to implement Pretty Understandable Democracy and run a trial election. In order to implement components by a rigorous separation of duties, we decided to implement components by group-wise student projects within a computer science class at the Technische Universität Darmstadt, Germany. In this paper, we present several improvements and refinements made to the original scheme. Thereafter, we report on our experience about the implementation of the revised scheme and running a trial election among the different development teams (each team being responsible for one component).
Structure. The remainder of this work is structured as follows: Section 2 reviews related work in the area of code voting. Section 3 summarizes the original PUD scheme. In Section 4, we outline the settings provided to the class. In Section 5, we present several improvements of the original PUD scheme. Section 6 is dedicated to the implementation process. Section 7 reports on our lessons learned from conducting the trial election. Section 8 concludes this work and provides guidance for future research.

2 Related Work

Chaum's seminal work on code voting [Cha01] has motivated many researchers to build their schemes upon the same idea [JFR13, JRF09, JR07a, JR07b, JRF10, Hel09, HS07, HSS08]. None of these schemes follows the distribution of trust as rigorously as Pretty Understandable Democracy (PUD) does. While some of the proposed schemes do not provide receipt-freeness (hence, a malicious voter can violate secrecy of the vote) [Cha01, JFR13, JRF09], other schemes build upon dedicated trustworthy hardware [JR07a, JR07b, JRF10] or single voting servers [Hel09, HS07, HSS08] to ensure secrecy and/or integrity of the vote. The Norwegian Internet voting system [iEGT12] also uses some kind of code voting. While their verification code approach prevents single components from undetectably violating integrity, secrecy builds upon the assumption of a trustworthy voter platform [SVK12, KLH13] . The only scheme we are aware of following the distribution of trust principle as precisely as PUD is Pretty Good Democracy (PGD) [RT09][2]. As opposed to PGD, PUD is tailored towards understandability and therefore real-world applicability.

3 Pretty Understandable Democracy

This section provides a brief overview of Pretty Understandable Democracy (PUD) [BNOV13], its code sheets, the different entities, and the different phases (setup phase including auditing, voting phase, and tallying phase).

[2]It should be emphasized that PGD's adversary model is stronger because stored-as-cast integrity can be increased linearly with number of trustees, while PUD allows further conspiracies to violate integrity.

3.1 Code Sheets

Similar to other code voting schemes, code sheets are distributed using an out-of-band channel such as the postal service. Furthermore, voters get different code sheets. Different from the code sheets in other schemes, the PUD code sheet is divided into three parts: The first part consists of a permuted list of candidates[3]. The permutation differs from code sheet to code sheet. The second and third parts consist of codes which are concatenated and used to cast a vote for a specific candidate. The respective code sheets parts are associated by a shared index. A typical PUD code sheet is depicted in Figure 1. Note, different from the original code voting approach from Chaum [Cha01], the code sheet contains one additional code which is used throughout the voting phase to acknowledge the reception of a valid voting code[4].

i	i	i
Dave	$d8li$	$03gh$
⋮	⋮	⋮
Bob	$h68g$	$mlp4$
–	$y7rt$	$82g5$

Figure 1: Code sheet in PUD

3.2 Entities

The voting system consists of five different entities and a variable number of trustees. Their roles and duties are:

- *Trustees (T)* generate the election key pair in a distributed manner, i.e. they agree on a public key and each trustee knows one share of the corresponding key. The underlying cryptosystem must provide additive homomorphism, hence exponential ElGamal is used. Furthermore, the key generation process is tailored towards providing a trade-off between secrecy and robustness [Ped91]. They are involved in the auditing and tallying phase as in both phases information needs to be decrypted (while this is done in a distributed manner).
- *Registration Authority (RA)* generates, in the setup phase, the first part of the code sheets, i.e. the permuted candidate list. During the voting phase, RA is responsible for the election website.
- *Voting Authority 1 (VA1)* and *Voting Authority 2 (VA2)* generate, in the setup phase, the second and third part of the code sheets. They are also involved in the voting phase.
- *Bulletin Board (BB)* All data necessary to verify (or audit) is published on the BB. Therefore all entities except DA have write access and everyone (including voters and the public) have read access.
- *Distribution Authority (DA)* has a key role in the setup phase: It receives the different code sheet parts, put them in an envelope, provides those that should be audited, and distributes the remaining envelopes to the voters.

There are two further entities involved in the election processes, namely the *voter*, who is a person eligible to vote, and the *voter's platform* which is used by the voter to cast his/her vote.

[3]Note, according to [DHR+11] permutations of candidates are in general legally compliant and in particular for simple ballots such as n out of m candidates.

[4]In [Cha01], there is one acknowledge code per candidate which might serve to prove an individual vote. Therefore, a number of schemes propose to only have one acknowledge code per code sheet.

3.3 Setup Phase

The setup phase consists of the following sub-phases: key generation, code sheet generation, committing on these code sheets, auditing, anonymizing and distributing them to eligible voters. These are explained in the following paragraphs.

3.3.1 Key generation.

Trustees generate the election key pair. All authorities generate RSA key pairs for TLS, which will be signed by a valid CA afterwards.

3.3.2 Code sheet generation.

The generation of the three code sheet parts is done distributively and secretly. Each code sheet part is assigned to an index to enable *DA* to put the proper parts in one envelope. For each index, *RA* generates a permutation and re-orders the canonical candidate list according to that permutation. For each index, *VA1* and *VA2* independently generate random codes for all candidates and the acknowledge code. All codes generated by one entity have to be unique which is to ensure integrity and differ for the two entities to avoid confusion. After an authority generated code sheet parts for all indices, the respective code sheet parts are put in envelopes and their index is printed on these envelopes. *RA*, *VA1*, and *VA2* forward their indexed envelopes to the *DA*. In order to allow code sheet auditing, more code sheets than needed are generated throughout the code sheet generation.

3.3.3 Committing on code sheets.

Afterwards, *RA*, *VA1*, and *VA2* commit on the respective code sheet parts by publishing encryptions (using the election key) of the permuted candidate list (each candidate is encrypted separately) and the generated codes (all codes per index are encrypted together).

3.3.4 Auditing code sheets.

During the auditing process, indices are randomly chosen by the *Trustees*. For each announced index the three code sheet parts stored at *DA* are opened. Furthermore, the encrypted data for the specific index is downloaded from the *BB* and decrypted by the *Trustees*. The public can verify that the committed code sheet data and the printed code sheets part match. Audited code sheets are discarded.

3.3.5 Anonymizing and distributing code sheets.

All remaining code sheets sharing the same index are put into neutral envelopes. These are put in a box and then shuffled. After this anonymization step, the neutral envelopes are distributed to the eligible voters.

3.3.6 Setting-up the servers.

As *RA*, *VA1*, and *VA2* are also involved in the voting phase as well as the *BB*, corresponding servers are set up. Furthermore, the election register is loaded on *RA*'s server component and *VA1* and *VA2* download the information posted by *RA*.

3.4 Voting Phase

In order to cast a vote, the voter visits the election website, which is hosted by *RA*. Afterwards, the voter authenticates herself using strong authentication such as a national electronic ID. Then, the voter submits the code, which corresponds to his/her candidate. Referring to Figure 1 if the voter intends to cast a vote for Bob, he/she submits the concatenated code $d8li03gh$. *RA* divides

these codes into the two respective parts and sends the first part to *VA1* and the second one to *VA2* (*d8li* to *VA1* and 03*gh* to *VA2* in the example). Both entities generated their respective code sheet parts of all voters. Consequently, both authorities are able to deduce for which code sheet index i at which position p the code part was generated. Both components check whether the code was used before and whether it is a valid code. If all checks are successfully passed, *VA1* and *VA2* independently re-encrypt the encrypted candidate for the corresponding code sheet i and position p (without knowing the plaintext candidate); the encrypted candidate corresponding to code sheet i and p has been published throughout *committing on code sheets* step. They both sign their respective re-encrypted candidate and send this information to the *BB*. After receiving this data, *BB* publishes it and sends each *VA1* and *VA2* a confirmation back. After receiving the confirmation, both *VA1* and *VA2* forward their part of the acknowledgement code to *RA*, which concatenates the acknowledgement codes and sends it to the voter. The voter checks whether the acknowledgement code matches the one on his/her code sheet.

3.5 Tallying Phase

To tally the votes, *RA* signs the total number of authenticated voters, which have participated in the election and sends this information to *BB* and the *Trustees*. The *Trustees* request the signed re-encrypted candidates from the *BB*. *BB* returns both list (one from *VA1* and one from *VA2* with the voting phase. After this, the *Trustees* check whether the number of entries in the list matches the number of authenticated voters. Then they sum up the entries in each list homomorphically and decrypt both resulting sums in a distributed manner (including correctness proofs for the decryption). For each decrypted sum, the *Trustees* solve the discrete logarithm and compare if the both resulting values match. If so, this value is declared to be the election result. The result is published on the *BB* along with the zero knowledge proofs of the correct decryption.

3.6 Security Model

PUD ensures secrecy and integrity (encoded-as-intended, cast-as-encoded, stored-as-cast, and tallied-as-stored) under the assumption that coercion does not take place, cryptographic primitives cannot be broken, voters do not forward authentication material due to the strong authentication in place, and do not fall for phishing. In addition, it is assume that the adversary cannot corrupt more than one entity from the set of authorities, voters, or voter's platforms. Given that adversary model, PUD defends the security requirements against a corrupted voter, a corrupted platform, and a corrupted authority. A more thorough review of the related work can be found in [BNOV13].

4 Settings

Pretty Understandable Democracy (PUD) has been implemented within a student project as part of the lecture *Electronic Voting* in the winter term 2013/14 at the Technische Universität Darmstadt, Germany. Students participating in this course had a background in computer security and cryptography.

4.1 Pre-considerations

Before the course started, it was identified which parts should be realized and which are not realistic within a course exercise. First, we simplified the authentication step during the election process by simply using the voter's name instead of a strong authentication method. This pseudo-authentication could be easily replaced by a strong authentication method like the German electronic ID card if this or a similar implementation would be used in a real-world election.

In PUD, any communication between two components is secured by applying TLS. In contrast to a real-world system, the project management team signed the public key for each component and acted as a Certificate Authority.
In addition, it was decided that the servers did not have to be protected against hackers etc.. In a real-world scenario protection against several threats, like denial of service attacks (DoS), would be necessary but was out of scope for the implementation task. However, for the course, this enabled the students to use their own laptops.
Motivated by a newspaper report[5] we decided to tailor our trial election towards the State Election (German: *"Bürgerschaftswahl"*) of the Hanseatic City of Lübeck and implemented the respective ballot from the last state election. Furthermore, it was decided that $35 - 40$ voters (i.e. all students and supervisors) should be eligible to vote in the trial election at the end of the semester.
The software development teams were free to choose any programming language, as long as they are able to provide communication interfaces for the other components. This has several advantages: First, due to the different programming skills within specific languages, students could build upon their preferred languages. Second, relying on one single programming language could result in system vulnerabilities due to the compiler. An adversary could corrupt the whole system by just corrupting the used compiler. By using different programming languages also different compilers/interpreters are used. Assuming an adversary can corrupt one compiler/interpreter just one component would be corrupted but not the whole system.
For distributed key generation and tallying (see Section 3) we extended an already existing android app [NKMV13] which was already able to perform distributed ElGamal key generation. For the Trustees' secret sharing, we defined a threshold of two out of three.

4.2 Organization

There were several software development teams (each one consisted of 2 to 3 students) while each team was assigned to one component and one phase. There were the following software development teams: *VA1*-setup, *VA1*-voting, *VA2*-setup, *VA2*-voting, *Trustees*-audit, *Trustees*-tallying, *RA*-setup, *RA*-voting. In addition, there were the project management team, the *BB* team, and the *DA* team. Students in the software development team were explicitly told to not copy any code from other groups to ensure the required separation of duty (SoD).

4.3 Schedule

The lecture started on October 18, 2013. There were two sessions to discuss the PUD scheme. The group assignment was done afterwards. Correspondingly, the software development part started on November 5th, 2013 and the trial election was scheduled for February 7th, 2014. Thus, the teams had about three months time to implement and test their components.

4.4 Project management

The software development teams were asked to send the project management team their component design, their interfaces and their project schedule until November 15th, 2013. This was done in order to detect and correct design flaws in an early stage of the development process. As target date for the first integration test, the project management team proposed January 15th, 2014. During the development process the software development teams were rather free to organize themselves, but they were repeatedly asked to report their current status to the project management.

[5] `http://www.segeberger-zeitung.de/Schleswig-Holstein/Landespolitik/Kommunalwahl-2013/Albig-erwaegt-Online-Wahl`

5 Protocol Refinements

After foundational concepts of electronic voting has been introduced to the students, there were two lectures on Pretty Understandable Democracy in which the scheme was introduced and discussed with the students. During these discussions, a couple of improvements have been identified. These are proposed and discussed in this section.

5.1 Candidate encoding

The original proposal was to encode candidates within one single ciphertext. Due to the fact that throughout the tallying process, all encryptions are summed up, each individual encryption of a candidate must also encode *null* encodings of all other candidates. As a consequence, computing the discrete logarithm for such a complex encoding results in a computationally-intensive task even for small-scale elections. Following the multi-candidate punch-hole vector-ballot by Kiayias and Yung [KY04], our revised scheme encodes each candidate into a separate encryption indicating whether the candidate is selected or not. Therefore, the revised scheme encodes each candidate into a separate encryption indicating whether the candidate is selected or not. Therefore C encrypted blocks are sent where C is the number of candidates. Each block has the form $\{g^x\}^r_{pk_T}$ where r is a random number and x is the number of votes for this candidate. If the voter has exactly one vote this is either 1 or 0. For example there are 3 candidates and the voter votes for candidate 1 and 3. The corresponding encodings are (g^1, g^0, g^1) and the respective encryptions are given as $(\{g^1\}^{r_1}_{pk_T}, \{g^0\}^{r_2}_{pk_T}, \{g^3\}^{r_3}_{pk_T})$. Due to this improvement the necessary number of re-encryptions is increased to C for each voter. Furthermore during the tallying process $2 \cdot C$ homomorphic sums are calculated. To overcome these drawbacks compared to the encoding in [BNOV13] the tallying performance is improved. The encrypted homomorphic sums for each candidate are given as $g^{c_1}, g^{c_2}, ..., g^{c_n}$ where c_i describes the number of votes for candidate i. To solve g^{c_i} the discrete logarithm problem has to be solved but the number of necessary modular exponentiations to find all c_i is limited to $\sum_{i=1}^{C} c_i \leq V$ modular exponentiations where V is the number of eligible voters. This is solvable by using bruteforce. Compared to up to $V \cdot 10^{(C-1)\cdot\lceil\log_{10}(V)\rceil}$ modular exponentiations which are necessary to tally as described in [BNOV13] this is a significant improvement.

5.2 Cross-checking indices and positions

Originally, PUD prescribed the following procedure: After *RA* split the voting code apart and forwarded the respective parts to *VA1* and *VA2*, *VA1* and *VA2* independently re-encrypt the ciphertext related to the specific voting code (over index and position of the voting code). It turns out that a malicious voter might however prevent the computation of an election result by submitting code parts that represent different candidates, e.g. on the middle code sheet part, the voter would chose the code at position 3 and at the right code sheet part, the voter chooses the code at position 4. In such a case, *VA1* and *VA2* would re-encrypt different candidates and the computed homomorphic sum of both authorities would differ. Therefore, in addition to validity checks, *VA1* and *VA2* cross-check that they obtained codes of the same index and the same position. In case the code is invalid or a mismatch is detected, *VA1* and *VA2* log the corresponding request and inform *RA* that informs the voter.

5.3 Code length

The PUD scheme builds upon the use of voting codes to ensure the conduction of secure election. The length of these codes plays a substantial role to the scheme because it directly impacts security and usability of the scheme. In the final part of this section, we therefore analyze which length voting codes shall have. In order to have unique codes, for C candidates and V voters, there are at least $(C + 1) \cdot V$ codes per *VA* required. To allow a sufficient proportion of the code sheets to be randomly audited, a factor λ is used. Therefore $\lambda \cdot (C + 1) \cdot V$ codes are needed for each

VA. Furthermore, the codes generated by *VA1* and *VA2* are disjoint which results in a factor 2 of generated codes[6]. Therefore $2 \cdot \lambda \cdot (C+1) \cdot V$ codes are needed for both *VA*s. This means that $\log_2(2 \cdot \lambda \cdot (C+1) \cdot V)$ bits are necessary for each code to ensure that all codes are different. For the trial election, we set $\lambda = 2$. The number of candidate was according to the proposed ballot $C = 14$ and the number of voters $V = 50$ (because of the number of participants). This takes $\lceil \log_2(2 \cdot 2 \cdot (14+1) \cdot 50) \rceil = 12$ bits. With `Base32` encoding, each code consists of 3 characters. In order to prevent guessing attacks launched by *RA*, there must be more characters within each code. We propose to extend each code by one further character. Note, as 50 voters is a rather small number, we were interested to study the practicability of PUD for larger amount of voters (while keeping $\lambda = 2$): For 500.000 voters, 25 bits (5 characters) are needed; for 1.000.000, 26 bits (6 characters) are needed. Even for 5.000.000 just 29 bits (6 characters) are needed. From this we are able to conclude that code lengths scale well with the number of voters.

6 Implementation

In this section, we explain the programming interface, the different languages in place, the user interfaces, and the way the prototype was tested.

6.1 Programming Interfaces

In order to ensure a smooth communication between the involved entities, the students agreed on a REST API to receive and send data. To publish the specific syntax for each command an internal Wiki was used in which each team documented all available commands for their API. Some students did never work with a REST API and had to start learning it first.

6.2 Used Programming Languages

As programming languages Python, Java and Scala are used. Both parts of *RA* and *BB* are written in Python, both parts of *VA1* and *VA2* are written in Java and the *DA* is written in Scala.

6.3 Election Material and User Interfaces

The election materials as well as the user interfaces were developed in an iterative process, i.e. members of different teams provided feedback as well as friends not being involved in the process.

6.3.1 Election material

The election material was developed by the *DA* team in close collaboration with the *RA*-voting team. The invitation letter is the same as the interface shown in Figure 5 and an example code sheet from *VA1* is shown in Figure 2.

6.3.2 Vote casting interfaces

There are in total five relevant interfaces. Note, we translated the interfaces for this paper from German to English. Once visited the election website, information about the Internet voting process is displayed (see Figure 3). In order to proceed, the voter needs to click on 'Authenticate now'. The voter, then, authenticates himself/herself using the interface provided in Figure 4. As outlined in section 4, the voter only has to write his/her name, rather than authenticate by strong authentication means.

[6] To realize this, *VA1* could generate only those codes that start with 0 and *VA2* respectively generates only codes that start with 1.

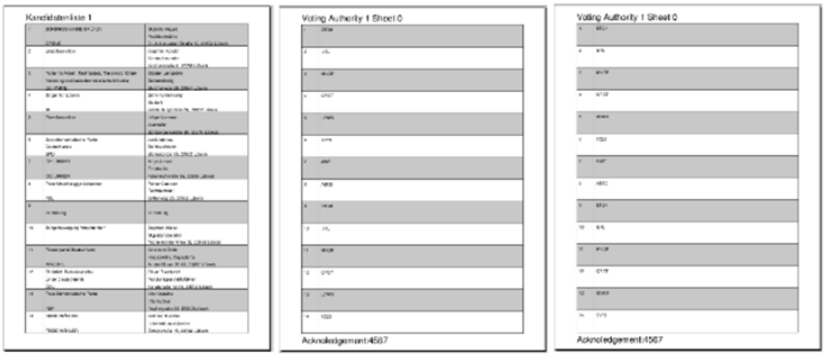

Figure 2: Code sheet of our trial election.

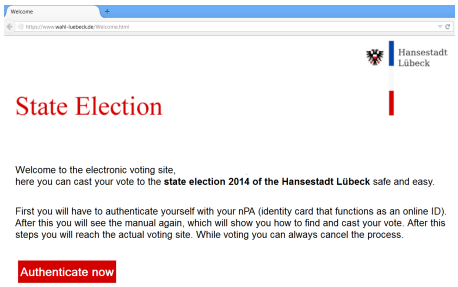

Figure 3: Welcome interface.

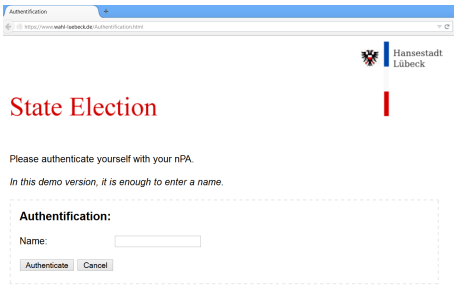

Figure 4: Interface to authenticate.

After being authenticated, the next interface displays the election manual (similar to the election material received together with the code sheets)(see Figure 5). The students decided to provide this information here again as the vote casting process differs from what voters might expect and, thus, to ensure that voters who did not carefully read the election material are provided with a short summary. The voter continues by clicking on 'Vote now'.
The system re-directs the voter to the next interface on which he/she casts his/her vote (Figure 6). Both codes of his/her preferred candidate need to be provided in the field next to 'Vote'. Spaces will be deleted by the interface. The vote casting can either be completed by clicking on 'cast' or canceled. If the voter enters an invalid concatenated code, e.g. code 1 corresponds to candidate A and code 2 corresponds to candidate 2, an error message will be returned by the *RA*. Once cast, the interface displays the information that the vote has been successfully cast and the respective acknowledgement code as shown in Figure 7.

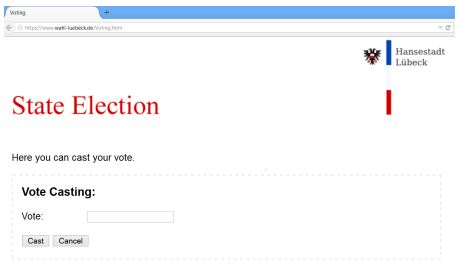

Figure 6: Voting casting interface.

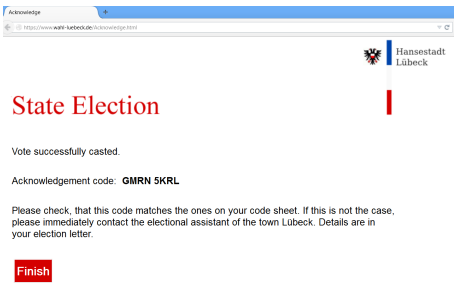

Figure 7: Interface with acknowledgement.

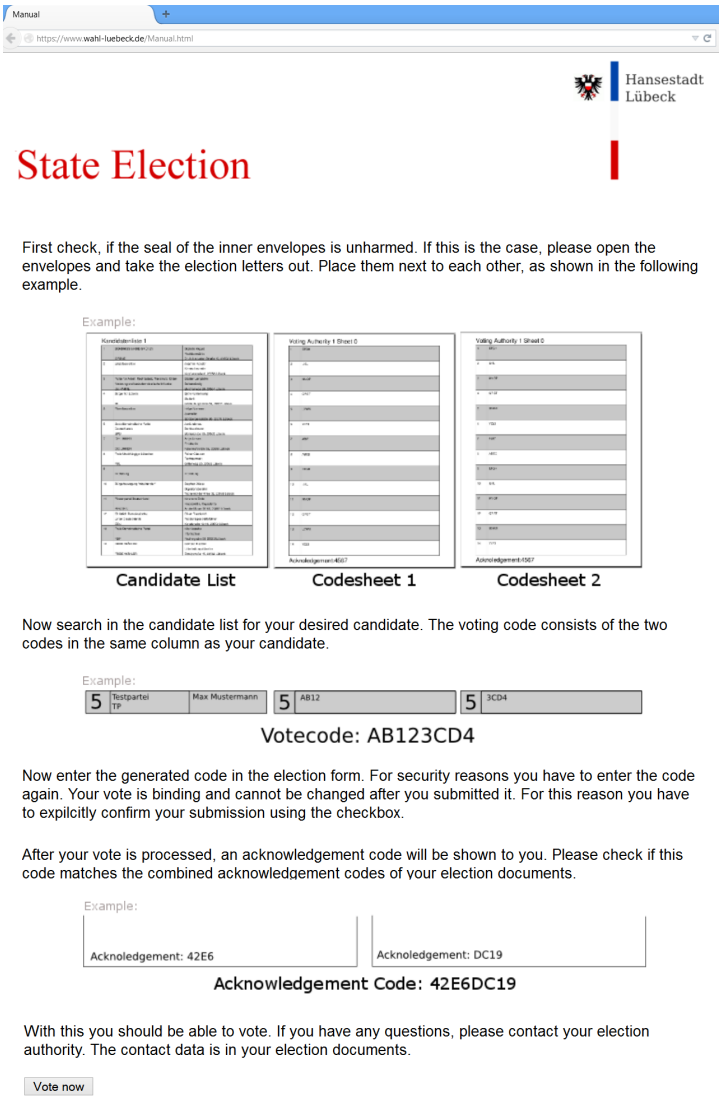

Figure 5: Interface with information about the election process.

6.3.3 Bulletin Board interfaces

The BB provides different sectors for all phases of the election process. Every entity has read access and except the Distribution Authority also write access. All data published on the Bulletin Board is signed by the publishing authority. For example, throughout the setup phase, commitments of code sheets are published on the BB (see Figure 8). The announced election result at the end of the tallying phase is shown in Figure 9.

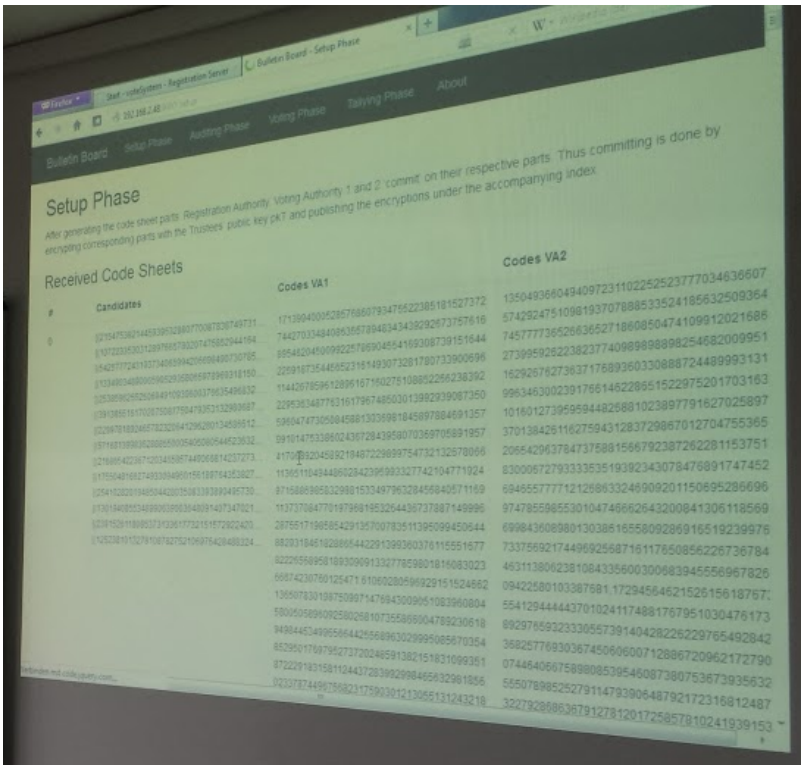

Figure 8: Committed and encrypted parts of the code sheets.

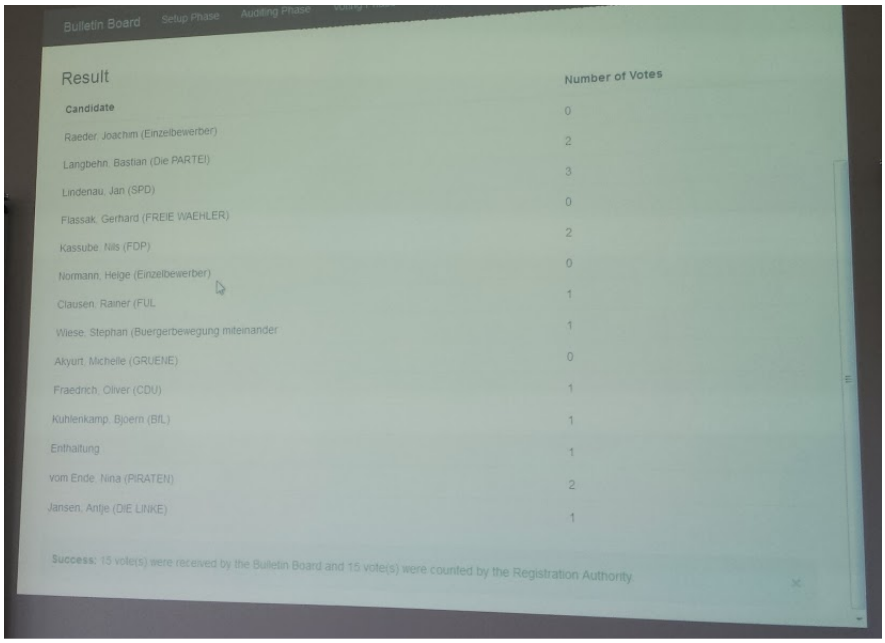

Figure 9: The end result of the election is shown on the BB.

6.4 Tests

To test their components the teams wrote their own test cases and some of them set up test servers which respond static messages depending on the request with the syntax described in the internal Wiki. With such servers the teams could also test the communication with other components.
Unfortunately, some teams did not stick to the plan on the first test, which was as announced on January 15th. Therefore, the final complete test took place at February 6th, 2014, only one day before the trial election. At the final test some problems occurred, which had to be fixed: The communication from any component to *VA1* did not work because of a TLS error. Furthermore the tallying module did not work properly because the group did not implement homomorphic tallying properly. To fix the communication and tallying problems, the students worked until late night and the whole morning before the trial election.
This experience shows that time schedules are even more important if (voting) systems are developed in such a distributed manner.

7 Lessons Learned from Running the Trial Election

The trial election was conducted on February 7th, 2014. Assembling all the needed papers (three code sheets and the election manual) took about 20 minutes (with one printer) for the small trial election with 50 voters, where ten persons in parallel took care of preparing the voting papers. This process could possibly be improved by special machines. Even without machines, the process could be organized in a way that is acceptable as in many German cities the postal voting material is also prepared manually.
Auditing only five code sheets took us more than 10 minutes. It just takes time to open the envelopes and read aloud all the candidates, then all the codes from *VA1* and then all the codes from *VA2* for each audited code sheet. It even takes more time, if this is done in a transparent manner, i.e. the present observers can follow the process.
When entering the codes, we noticed that some participants were confused by entering both parts of the code in the same text field. It might be worth providing two different fields in future and clearly indicating which code to enter in which field. The different views of the bulletin board were clear to the participants. However, it was also discussed that in case - due to transparency requirements - it is assumed that also voters should understand the content of the bulletin board, further information needs to be provided.

8 Conclusion

The present work reports about the experience of refining and implementing Pretty Understandable Democracy (PUD) and running a trial election with that scheme as part of a computer science course. Overall, the implementation and the election have been a success. The insights gained throughout the implementation and the trial election process are manifold and serve as guidelines for future research.
PUD has been introduced as a theoretical concept and as such several details remained open. This gap forms the motivation for the present work. The first refinement is the multiple ciphertext encoding of single votes, which reduces the number of modular exponentiations needed throughout the tallying process significantly. In order to prevent malicious voters from blocking the calculation of the election result, the voting authorities cross-check the consistency of voting codes. Furthermore, we analyzed the required lengths of voting for different election settings. Finally, in order to conduct the trial election as close as possible to real-world elections, we proposed user interfaces tailored towards the state election of the Hanseatic city of Lübeck which currently considers introducing Internet voting as new voting channel. The contributions of this work builds *one* step towards PUD's real-world applicability knowing that there are many challenges open challenges before its first usage.

For the future, we see space for improvements both from the conceptual and from the implementation perspective. Throughout the trial election, individual code sheet parts had to be combined into one envelope and sent out to voters. This results in significant organizational and time-intensive effort. We consider revising the code sheet distribution process, thereby lowering the organizational effort. Discussions among the students and the staff show that from a usability perspective the scheme is going into the right direction. In order to evaluate the scheme's usability in an unbiased manner, user studies will be conducted in the near future. PUD has been tailored towards a trade-off between security and transparency. Nevertheless, the scheme builds upon several cryptographic primitives. We plan to investigate the scheme's understandability by preparing information and education material and evaluating it in user-studies.

Acknowledgment

This work has been developed within the project ComVote, which is funded by CASED.

References

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[BNOV13] Jurlind Budurushi, Stephan Neumann, Maina Olembo, and Melanie Volkamer. Pretty Understandable Democracy - A Secure and Understandable Internet Voting Scheme. In *8th International Conference on Availability, Reliability and Security*, pages 198–207. IEEE, 2013.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha01] David Chaum. Sure vote: Technical overview. In *Proceedings of the Workshop on Trustworthy Elections (WOTE 01)*, 2001.

[DHR+11] Denise Demirel, Maria Henning, Peter Y. A. Ryan, Steve Schneider, and Melanie Volkamer. Feasibility analysis of prêt à voter for german federal elections. In *VOTE-ID*, pages 158–173, 2011.

[Hel09] Jörg Helbach. Code Voting mit prüfbaren Code Sheets. In *GI Jahrestagung*, pages 1856–1862, 2009.

[HS07] Jörg Helbach and Jörg Schwenk. Secure Internet Voting with Code Sheets. In *VOTE-ID*, pages 166–177, 2007.

[HSS08] Jörg Helbach, Jörg Schwenk, and Sven Schäge. Code Voting with Linkable Group Signatures. In *Electronic Voting*, pages 209–208, 2008.

[iEGT12] Jordi Barrat i Esteve, Ben Goldsmith, and John Turner. International experience with e-voting. 2012.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *ACM Workshop on Privacy in the Electronic Society*, pages 61–70. ACM, 2005.

[JFR13] Rui Joaquim, Paulo Ferreira, and Carlos Ribeiro. EVIV: An End-to-end Verifiable Internet Voting System. *Computers & Security*, 32:170–191, 2013.

[JR07a] Rui Joaquim and Carlos Ribeiro. CodeVoting: Protecting Against Malicious Vote Manipulation at the Voter's PC. In *Frontiers of Electronic Voting*, 2007.

[JR07b] Rui Joaquim and Carlos Ribeiro. CodeVoting Protection Against Automatic Vote Manipulation in an Uncontrolled Environment. In *VOTE-ID*, pages 178–188, 2007.

[JRF09] Rui Joaquim, Carlos Ribeiro, and Paulo Ferreira. VeryVote: A Voter Verifiable Code Voting System. In *Proceedings of the 2nd International Conference on E-Voting and Identity*, VOTE-ID '09, pages 106–121. Springer-Verlag, 2009.

[JRF10] Rui Joaquim, Carlos Ribeiro, and Paulo Ferreira. Improving Remote Voting Security with CodeVoting. In *Towards Trustworthy Elections*, pages 310–329, 2010.

[KLH13] Reto E Koenig, Philipp Locher, and Rolf Haenni. Attacking the verification code mechanism in the norwegian internet voting system. In *E-Voting and Identify*, pages 76–92. Springer, 2013.

[KOKV11] Fatih Karayumak, Maina Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios - an open source verifiable remote electronic voting system. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*, 2011.

[KY04] Aggelos Kiayias and Moti Yung. The vector-ballot e-voting approach. In *Financial Cryptography*, pages 72–89. Springer, 2004.

[LSBV10] Lucie Langer, Axel Schmidt, Johannes Buchmann, and Melanie Volkamer. A taxonomy refining the security requirements for electronic voting: analyzing helios as a proof of concept. In *5th International Conference on Availability, Reliability and Security*, pages 475–480. IEEE, 2010.

[NKMV13] Stephan Neumann, Oksana Kulyk, Lulzim Murati, and Melanie Volkamer. Towards a practical mobile application for election authorities (demo). In *4th International Conference on e-Voting and Identity (VoteID13)*, 2013.

[OBV13] Maina M. Olembo, Steffen Bartsch, and Melanie Volkamer. Mental models of verifiability in voting. In *Proceedings of the 4th International Conference on E-Voting and Identity*, Vote-ID'13, pages 142–155, Berlin, 2013. Springer-Verlag.

[Pan14] Panda Security. Annual Report Pandalabs 2013 summary. `http://press.pandasecurity.com/wp-content/uploads/2010/05/PandaLabs-Annual-Report_2013.pdf`, 2014. Online; accessed 30 May, 2014.

[Ped91] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT'91*, pages 522–526. Springer, 1991.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty Good Democracy. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols Workshop*, pages 111–130. Springer, 2009.

[SVK12] Oliver Spycher, Melanie Volkamer, and Reto Koenig. Transparency and technical measures to establish trust in norwegian internet voting. In *E-Voting and Identity*, pages 19–35. Springer, 2012.