Pretty Understandable Democracy 2.0

Stephan Neumann, Christian Feier, Perihan Sahin, and Sebastian Fach
Technische Universität Darmstadt / CASED, Germany
Email: stephan.neumann@cased.de, feier@rbg.informatik.tu-darmstadt.de,
perihansahin87@hotmail.com, info@sebastian-fach.de

***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.**

I. 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 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 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. pre-auditing 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).

Related Work. Chaum's seminal work on code voting [Cha01] has motivated many researchers to build their schemes upon the same idea, e.g. [HS07], [JRF09], [RT09]. 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 [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. A more thorough review of the related work can be found in [BNOV13].

PUD in a Nutshell. Code sheets in PUD have three parts: The first part consists of a permuted list of candidates, the second and third parts consist of random and unique codes. The code parts each hold one further code which corresponds to an acknowledgement code. Throughout the code sheet generation, the respective authorities commit on their generated code sheets by encrypting them with an additively homomorphic encryption scheme (in our case ElGamal) and publishing the code sheet parts on a bulletin board. Before randomly

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

[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.

sending out composed code sheets to voters, a fraction of code sheets is audited by comparing the printed code sheets to the encrypted version on the bulletin board. Once, the voter casts the concatenated code (from the second and third code sheet parts) which corresponds to her preferred candidate, the code parts are forwarded to the authorities that generated the respective parts. Given the encryptions of code sheet parts, both authorities are able to re-encrypt the candidate ciphertext that corresponds to that code *without* knowing the candidate within that ciphertext. In the tallying phase, the published candidate re-encryptions are summed up homomorphically and distributively decrypted. By calculating the discrete logarithm, the final result can be obtained. The tallying process is publicly verifiable.

Remark. The full version of this paper [NFSF14] contains an extended introduction to the PUD scheme and all user interfaces. For a detailed review of PUD's security model, we refer the reader to the original PUD publication [BNOV13].

II. Preliminary Setting and Task Organization

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.

1) Pre-considerations: Before the course started, it has been agreed on 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. 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. 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, this enabled the students to use their own laptops. Motivated by a newspaper report[3] we decided to tailor our trial election towards the *"Bürgerschaftswahl"* (which translates to State Election) 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 were able to provide communication interfaces for the other components. This had 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. For distributed key generation and tallying, we extended an already existing Android application [NKMV13]. We defined a threshold of two out of three.

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

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: Voting authorities (*VA1* and *VA2*) *VA1*-setup, *VA1*-voting, *VA2*-setup, *VA2*-voting, *Trustees*-audit, *Trustees*-tallying, the registration authority *RA*-setup, *RA*-voting. In addition, there were the project management team, the bulletin board (*BB*) team, and the distribution authority (*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).

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) Project management: The software development teams were asked to send their component design, their interfaces and their project schedule until November 15th, 2013 to the project management team. 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 free to organize themselves, but they were repeatedly asked to report their current status to the project management.

III. Protocol Refinements

After foundational concepts of electronic voting were 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 were identified. These are proposed and discussed in this section.

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 C encrypted blocks are sent where C is the number of candidates. Each block has the form $\{g^x\}_{pk_T}^r$ 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 $(\{g^1\}_{pk_T}^{r_1}, \{g^0\}_{pk_T}^{r_2}, \{g^3\}_{pk_T}^{r_3})$. 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 brute-force. 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.

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.

Code length. The PUD scheme builds upon the use of voting codes to ensure the conduct of secure elections. 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. 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 $\lambda = 2$. With `Base32` encoding, each code consists of 3 characters.

IV. Implementation

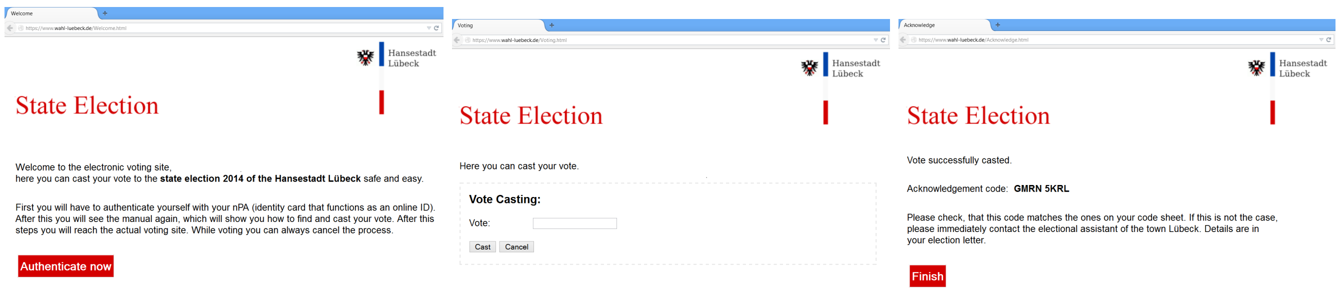
Programming Languages and Programming Interfaces. The development teams agreed on Python, Java and Scala as programming languages. Both parts of *RA* and *BB* are written in Python, both parts of *VA1* and *VA2* are written in Java and the *DA* is written in Scala. In order to ensure a smooth communication 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.

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. The election material was developed by the *DA* team in close collaboration with the *RA*-voting team. Once visited the election website, information about the Internet voting process is displayed (see Figure 1(a)). In order to proceed, the voter needs to click on 'Authenticate now'. The voter, then, authenticates himself/herself. After being authenticated, the next interface displays the election manual (similar to the election material received together with the code sheets). 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 1(b)). 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. Once cast, the interface displays the information that the vote has been successfully cast and the respective acknowledgement code as shown in Figure 1(c). 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

Tests. To test their components the teams wrote their own test cases. 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 correctly because the group did not implement homomorphic tallying properly. To fix these 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 a distributed manner.

V. Lessons Learned

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.


Welcome

https://www.wahl-luebeck.de/Welcome.html

Hansestadt Lübeck

State Election

Welcome to the electronic voting site,
here you can cast your vote to the **state election 2014 of the Hansestadt Lübeck** safe and easy.

First you will have to authenticate yourself with your nPA (identity card that functions as an online ID). After this you will see the manual again, which will show you how to find and cast your vote. After this steps you will reach the actual voting site. While voting you can always cancel the process.

Authenticate now

(a) Welcome Interface.

Voting

https://www.wahl-luebeck.de/Voting.html

Hansestadt Lübeck

State Election

Here you can cast your vote.

Vote Casting:

Vote:

Cast Cancel

(b) Vote casting interface.

Acknowledge

https://www.wahl-luebeck.de/Acknowledge.html

Hansestadt Lübeck

State Election

Vote successfully casted.

Acknowledgement code: **GMRN 5KRL**

Please check, that this code matches the ones on your code sheet. If this is not the case, please immediately contact the electional assistant of the town Lübeck. Details are in your election letter.

Finish

(c) Interface with acknowledgment.

Fig. 1. User-interface

VI. 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. 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. 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. 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.

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

[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.

[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.

[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.

[NFSF14] Stephan Neumann, Christian Feier, Perihan Sahin, and Sebastian Fach. Pretty understandable democracy 2.0. Cryptology ePrint Archive, Report 2014/625, 2014. http://eprint.iacr.org/.

[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.

[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.