JETS

The USENIX Journal of Election Technology and Systems

Volume 3, Number 2 • August 2015

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

JETS

The USENIX Journal of Election Technology and Systems

Volume 3, Number 2 • August 2015

JETS articles in this issue will be presented at the 2015 USENIX Journal of Election Technology and Systems Workshop (formerly EVT/WOTE).

ISBN 978-1-931971-256 ISSN 2328-2797

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

USENIX Journal of Election Technology and Systems (JETS)
Volume 3, Number 2 • August 2015
www.usenix.org/jets/issues/0302

From Error to Error: Why Voters Could not Cast a Ballot and Verify Their Vote With Helios, Prêt à Voter, and Scantegrity II

Claudia Z. Acemyan[1], **Philip Kortum**[1], **Michael D. Byrne**[1, 2], **Dan S. Wallach**[2]
[1]Department of Psychology, Rice University
[2]Department of Computer Science, Rice University
6100 Main Street, MS-25
Houston, TX 77005 USA
{claudiaz, pkortum, byrne}@rice.edu and dwallach@cs.rice.edu

ABSTRACT
The aim of this paper is to identify user errors, and the related potential design deficiencies, that contributed to participants failing to vote cast and vote verify across three end-to-end voting systems: Helios, Prêt à Voter, and Scantegrity II. To understand why voters could not cast a vote 42% of the time and verify that their ballots were cast and counted with the tested e2e systems 53% of the time, we reviewed data collected during a system usability study. An analysis of the findings revealed subjects were most often not able to vote with Helios because they did not log in after encrypting their ballot but before casting it. For both Prêt à Voter and Scantegrity II, failing to vote was most frequently attributed to not scanning the completed ballot. Across all three systems, the most common reason participants did not verify their vote was due to not casting a ballot in the first place. While there were numerous usability failures identified in the study, these errors can likely be designed out of the systems. This formative information can be used to avoid making the same types of mistakes in the next generation of voting systems—ultimately resulting in more usable e2e methods.

INTRODUCTION

Imagine a real, large-scale election in which nearly half of the voters cannot cast a vote. To make matters worse, many of these voters thought they cast a ballot that would be counted in the election tally so they did not take further action, such as asking for help. As for the voters who realized they could not figure out how to vote, some of them gave up, went home, and wanted to avoid voting in the future. An abysmal scenario like this could potentially result in altered election outcomes and disenfranchised voters, which is completely unacceptable.

While this scenario is made-up, it is not completely farfetched. In a previous paper (Acemyan, Kortum, Byrne, & Wallach, 2014), close to half of the participants (42%) could not cast a vote using the tested voter verifiable systems in a mock election (see Figure 1). As for the vote-verification process, even more participants could not verify their vote—an optional yet fundamental system feature that helps make sure that the end-to-end systems are accurate and transparent, while keeping individuals' votes private. If these systems—as tested—were deployed in a real election, it would not be surprising if similar outcomes resulted.

This previous usability paper by Acemyan et al. was summative. The purpose of the reported test was to assess the usability of three, representative e2e voting systems—Helios, Prêt à Voter (PaV), and the Rice implementation of Scantegrity II. Per ISO 9241-11, three measures were reported to assess system usability: efficiency (time to vote cast and time to vote verify), effectiveness (per-contest error rate and percentages of both cast votes and verified votes), and user satisfaction (System Usability Scale, SUS, rating for vote casting and vote verifying). Since these standard measures were used along with an established protocol to assess voting system usability, the paper was able to not only summarize the state-of-the-art of e2e systems at a given point in time, but also compare them to all of the previously tested voting methods. The advantage of conducting a summative assessment like this is that researchers are able to understand how the usability of various systems generally compares to one another (Rubin & Chisnell, 2008).

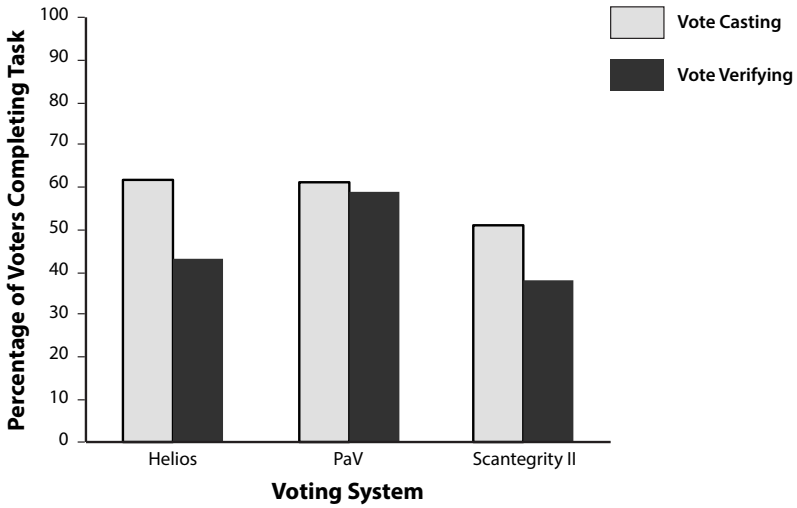


Figure 1. Percentage of voters completing vote casting and vote verification as a function of e2e voting system. These findings were reported in a previous paper (Acemyan et al., 2014).

Specific usability problems and their root causes were not reported in the previously published summative paper. Typically these types of findings are reported in a formative usability report. In a formative report, usability problems are identified so that they can be fixed in the next iteration of the systems (Sauro, 2010). Hence, the aim of this paper is to identify usability problems related to vote casting and vote verifying failures, since voting and verifying are two primary user goals. Concrete design suggestions that have the potential to address the most impactful usability impediments can aid in the development of future iterations of e2e systems, making them easier to use. While this method will not guarantee the identification and resolution of all system usability problems, it can likely improve the vote-casting and vote-verification completion rates in e2e systems.

Helios, PaV, and Scantegrity II have evolved since the study was conducted. The systems' development teams have likely even recognized some of the same usability failures reported in this paper and addressed them through alterations to system designs. Nonetheless, this formative usability assessment is still of value. It is still important for the voting research community to understand exactly why the systems failed as configured in this study so that developers of any voting system, or any party who selects and sets up voting methods for an election, can avoid making the same mistakes in the future.

METHODS

The participants, materials, procedures, and general research design were the same as those described in our previous paper (Acemyan et al., 2014). To summarize, 37 participants who were U.S. eligible voters participated in a mock election. This sample size was sufficient to identify usability problems related to vote casting and vote verifying. Research has shown that relatively small sample sizes can be used to identify the majority of usability problems (e.g., Lewis, 1994; Schmettow, 2012). Participation in the study entailed vote casting and vote verifying with each of three voter verifiable methods: Helios, Prêt à Voter, and Scantegrity II (all orders were used; participants were randomly assigned to an order). After using each system, participants completed surveys while keeping the particular system in mind. At the end of the study, participants completed a final general survey, were thanked, and debriefed.

If a participant struggled using a system or asked the experimenter for help, they were told, "Please do whatever you think you should do to cast your vote [or when verifying: check on or verify your vote]." Participants were not assisted during the usability test because then the usability of the system would have no longer been evaluated. Instead, the efficacy of the assistance would have become the object of evaluation. To withhold help during a usability test was to maintain equality across tested systems. If different types of assistance were offered for each system depending on its unique set of usability problems, then the internal validly of the study would have been

compromised. It would no longer have been clear whether the observed effects were due to the system design or the type and/or amount of assistance offered. For these reasons, in this study, we controlled for this variable by not offering any aid to participants. This decision aligns with strong usability protocols, which rarely allow for user assistance during the completion of tasks (Rubin & Chisnell, 2008).

A "control system" was not required in this standard usability assessment. As is standard in human factors usability research, the data regarding each tested system was sufficient as a metric to judge the usability of the evaluated system.

The Helios voting system and election was set up and run through Helios' website at vote.heliosvoting.org during the winter of 2013-2014. The tested version of PaV was developed by our research team. At the time this research study was conducted, the PaV system had not been developed to handle long ballots with numerous races like those found in the United States. Hence, a Rice version of PaV was developed based on published papers about PaV (e.g., Lundin & Ryan, 2008; Ryan et al., 2009; Ryan & Peacock, 2010; Ryan & Schneider, 2006), the PaV website (Prêt à Voter, n.d.), and in consultation with Peter Ryan. The tested variant of Scantegrity II was also put together by our team. The Scantegrity II systems used in the Tacoma Park elections were not available for usability testing. For this reason, a Rice version of Scantegrity II was developed based on printed materials used in the 2009 Takoma Park, Maryland election (Carback et al., 2010; www.takomaparkmd.gov), published articles about the system (e.g., Chaum et al., 2008; Chaum et al., 2009; Carback et al., 2010; Sherman et al., 2010), and email correspondence with both Aleks Essex and Richard Carback, researchers who have direct experience with the implementation (R. Carback, personal communication, December 13, 2012; A. Essex, personal communication, December 14, 2012). It should be noted that this study included only the Scantegrity II instructions available to voters during the actual use of the voting system because systems cannot rely on users to have engaged with optional instructions that are unavailable when voting with the method. When aspects of the system that might have the potential to impact usability were not specified—such as the exact scanner used and its setup—human factors principles were followed.

The critical incident technique was used to analyze the data. This method was selected because it is widely associated with engineering studies of human error; is useful as a rapid trouble shooting technique in initial, global system analyses; and has significantly contributed to the reduction of human error (Kirwan & Ainsworth, 1992).

Critical incident analysis involves, first, identifying events that have a major impact on system objectives, second, developing a taxonomy in which to categorize the events, and third, analyzing these events in order to identify the root causes of the issues so that system changes can be suggested in order to mitigate the problems and reduce errors (sometimes fixing these types of problems will require major alterations to the system design, not just tweaks to the existing system). The focus for change is on the equipment and system, not the users, as poor system designs contribute to errors and usability problems (Shneiderman, 1987).

This critical incident method of task analysis has been previously used in areas like pilot error and aircraft operations to change safety procedures and the design of equipment (e.g., Green, 1990), software documentation iterative design in order to ensure that the final documents met users' needs (Galdo et al., 1986), and medical errors in the contexts of reporting culture and safety performance to understand how hospitals differ (Itoh et al., 2010). Likewise, it can also be used to identify and categorize usability problems that led to voting system failures.

In the context of voting with e2e methods in this study, critical incidents are defined as a voter not being able to vote cast (goal 1) and/or vote verify (goal 2), two of the main system objectives. A user error is defined as the actions, whether intended or not, that prevented the voter from casting a ballot or from checking that their ballot had been cast. The system design deficiencies are the system features that likely contributed to the error occurrence. This paper does *not* attempt to address every possible usability struggle, slip, or mistake that might have occurred, or could occur, while vote casting and vote verifying; this list has the potential to be endless (Reason, 1990). For this reason, this paper focuses on these user errors that were deemed to be most serious.

To collect the data, the experimenter directly observed each participant use the three e2e systems. If a participant could not cast a ballot or check on their vote, the experimenter took notes about how the participant deviated from the procedures that were required to complete each task (see Appendicies A, B, and C for the typical steps that users go through to vote cast and vote verify with each system). When the data were analyzed, the errors were grouped by type. Then based on the resulting taxonomy and the notes on observed participant behaviors, the system design deficiencies that might have contributed to the behavioral errors were identified. System design deficiencies were identified by referring to resources that present best practices of interface design (e.g., Johnson, 2010; Sanders & McCormick, 1993; Shneiderman, 1987; and Tidwell, 2011).

The methods outlined in this section help to ensure external validity, or the generalizability of the study's findings to voting systems used in a real election. All systems have been tested in a well-controlled, laboratory environment to ensure that they were utilized under identical conditions. This type of environment of course differs from crowded polling stations filled with voters and election officials on election day. Nonetheless, if usability problems are found in a lab-based usability test, the problems should be fixed so that real voters also do not encounter them.

RESULTS

Helios

Vote Casting

As can be seen in Table 1, 14 (38%) participants were not able to cast a vote using Helios. The most frequent reason that voters did not cast a ballot is that **[A]** after encrypting their ballot, they did not log in with their email account to both verify their eligibility to vote in an election and have their most recent cast ballot counted in the final election tally. Twelve participants (32%) made this error. There are many design deficiencies that possibly contributed to this particular error, each of which are discussed in the following paragraphs.

[A1] *Voting system design is not aligned with users' prior voting mental models.* Figure 2 shows a screenshot of the user interface that the thirteen voters viewed when they announced to the experimenter that they had finished casting their ballot. At this point they had already specified their choices and reviewed them before being brought to the log in authentication screen. According to their prior voting mental model (likely based on typical voting procedures) they would be done voting—since authentication would usually occur before starting the voting process. Yet, with Helios, authentication had not yet taken place and needed to occur at this point in the vote-casting process. This mismatch between Helios' voting process and participants' mental models for typical voting procedures misguided subjects to conclude that they were finished, when in fact there was still more to do in order to vote cast (for more information about user mental models in interaction design, see Staggers and Norcio, 1993). For this reason it is essential for voters' mental models to align with the voting method that they are using, even if the e2e method is novel.

[A2] *Users are expected to read and process all text displayed on the interface*. Another potential design deficiency that could account for participants announcing prematurely that they were finished voting at the screen shown in Figure 2 is that this interface does not account for users who do not want to read large amounts of text. In this study, it appeared that participants did not read and/or cognitively process every piece of information shown on the webpage. Instead they likely saw the words, "We have received…," and concluded that they were done voting. At this point they thought their goal was fulfilled and did not put in further resources to make sure that it was actually achieved. The design of the interface did not support the user who scans pages, picking out individual words, instead of reading every word. Since a previous research study found that only 16% of their participants read web pages word by word (Nielsen, 1997), the text on this Helios page should be scannable, simple, and clearly worded. In addition, the most critical text should be the most prominent, and only the most essential information related directly to the particular step should be displayed.

Table 1. Errors and Likely Contributing Design Deficiencies that Led to Helios Vote-Casting and Vote-Verification Failures

Number of Participants (Percentage of Participants)	**Error**	**Contributing Design Deficiencies**
Vote Casting		
12 (32%)	Encrypted the ballot but did not log in with e-mail **[A]**	• Voting system design is not aligned with users' prior voting mental models **[A1]** • Users are expected to read and process all text displayed on the interface **[A2]** • Log in information is not entered directly on page **[A3]** • Too many steps are required to cast a vote **[A4]** • Incorrect organization of steps **[A5]**
1 (3%)	Did not press the cast button after a successful log-in **[B]**	• Users are expected to read and process all text displayed on the interface **[B1]** • Too many steps are required to cast a vote **[B2]**
1 (3%)	Refused to cast a ballot due to concerns about errors **[C]**	• Poor system usability degraded voter confidence **[C1]**
Vote Verification		
14 (38%)	Did not cast a ballot **[D]**	• See above vote-casting deficiencies
6 (16%)	Performed the wrong action **[E]**	• No instructions are provided for vote-verification process **[E1]**

Notes: This table includes observed user errors that led to a vote casting and/or vote verification failures. It does not include every critical incident observed throughout the study.

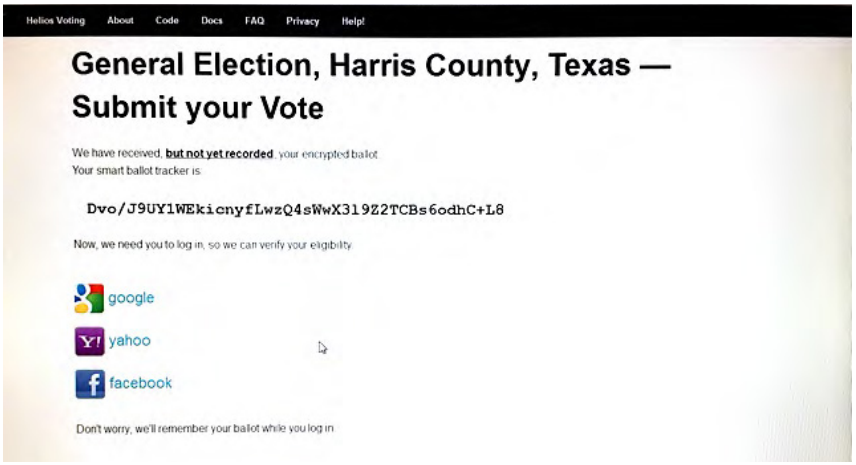


Figure 2. Helios screen that voters see after encrypting their ballot but before logging in so that they can later cast their ballot

[A3] *Log in information is not entered directly on page.* Another possible problem with the tested Helios system design is that when it is time to authenticate, the voters do not enter their email address and password directly on the page shown in Figure 2. Instead, voters are supposed to click the e-mail client icon, which takes them through another series of pages, and then returns them back to a page that allows them to finally cast their ballot. Due to this design, voters got lost in the voting process. Some participants did not have the appropriate knowledge required to successfully navigate the site (Spool, 1999, p.15). These subjects could not figure out how to log in because it was not apparent to them that they should click on one of the Google, Yahoo, or Facebook icons to proceed. Of those participants who managed to figure out this step, two participants navigated away from the Helios voting system and could never find their completed ballot again—ultimately giving up before casting their ballot. In this scenario, the site structure did not align with their behavior and consequent expectations (Spool, 1999). In addition they had to shift their attention away from their primary goal of vote casting to a new task of figuring out the voting interface. This shift can cause users to lose track of what they were doing, or where they were doing it (Johnson, 2010, p. 97). If, instead, participants could log in with their email credentials directly on the page shown in Figure 2 so that the process was more linear and streamlined, then there would be less of a chance that they would fail at logging in and/or become permanently lost in the process.

[A4, A5] *Too many steps are required to cast a vote* and *incorrect organization of steps.* Between the time the voter completes his or her ballot and then casts it, the person views at least five different screens (more if not already logged in). The developers of Helios were likely aiming to have the interface guide the user step by step in the prescribed order, which is generally considered to be a good interaction design. However, in the tested version of Helios, the number of steps makes the voting process feel drawn out. To make matters worse, the steps, accompanying information, and actions required on behalf of the system user are not chunked correctly. Particularly, the system does not strike a balance between the division of steps (and sub-steps) and the number of them, making the process so tedious that users look for the first instance that they might be done (i.e., seeing that the ballot was received), and then leave prematurely before the cast-ballot button ever gets pressed. To prevent fleeing voters (Everett, 2007), it would be best to keep both the distance short and organization of steps simple, yet meaningful, between the ballot completion page and the vote-casting page in order to make the stepwise process the most efficient, and effective.

[B, B1, B2] *Users are expected to read and process all text displayed on the interface* and *too many steps are required to cast a vote.* One participant (3%) failed to cast a ballot because they completed the log in process, but on the very last page did not press the final vote-casting button. One reason this error likely occurred was because on the final vote-casting screen there was too much text presented, some of which was confusing and not directly related to the task of pressing the button that casts the voter's ballot (refer to [A1] for more information on why this is a problem). The second reason that the error might have occurred has to do with there being too many discrete, cumbersome tasks required to vote—from ballot completion and review to multi-step encryption, logging in, and finally vote submission. Again, if some of the early steps were to be made invisible to the user (i.e., the system still includes the security mechanisms but does not require the voters to be aware of them and serve as an active participant in the process), then the number of cumbersome tasks and the associated errors might be reduced (for further information about this design deficiency, see [A3]).

[C, C1] *Poor system usability degraded voter confidence.* One participant (3%) refused to cast a ballot because they were concerned about errors. Specifically the user stated that they were worried both that the system could be making mistakes and that they were not using the system correctly. For these reasons, the participant wanted to talk to an election official to get help. Regarding their concerns about system errors, research should be conducted in the future to determine how to strategically increase a voter's confidence and trust in the system's reliability and accuracy. As for their concerns about whether or not they were performing the correct actions required to cast a vote, addressing system usability issues would improve the participant's confidence that they were using the system "correctly" and that the intended selections would be cast and counted accurately.

Vote Verification

In the vote-verification process, 14 (38%) participants were not able to check on their vote because they **[D]** did not cast a ballot (refer to Table 1). In other words, after subjects thought they voted, they could not do anything else with the system because there was not a cast ballot to check on. To correct this problem, the system needs to be redesigned so that every voter can both easily cast a ballot and recognize if they have or have not done so.

Of the 23 participants who were able to cast a ballot, six (26%) were not able to verify due to the following possible design deficiencies:

[E, E1] *No instructions are provided for the vote-verification process.* Another reason that voters were not able to check on their ballots with Helios is that they performed the wrong actions. Six participants encountered this type of error because vote-checking instructions were not provided to voters.

In this study, before the participants started to use any of the tested systems, the experimenter directed participants to first vote, then verify their vote. For this reason, participants knew vote verification was possible, somehow—an advantage real voters might not have. Since there were no vote-verification instructions and cues, there were many actions participants performed to achieve what they thought was a verification, when in fact it was not: participants viewed the review screen (see Figure 3) and then "googled" their smart ballot tracker (and found nothing), and/or they printed the smart ballot tracker from the review screen. Some participants had no idea what to do, and so they did not take any further action. To reduce the frequency of these mistakes, the system should provide voters with simple, concise vote-verification instructions both on the final vote-cast confirmation screen and through the vote-confirmation email. It would also be beneficial to provide simple, inline instructions throughout the verification process.

Of the 16 (43% of total participants) who were able to complete some form of vote verification, only half of these subjects completed verification procedures as outlined in the published papers describing Helios (e.g., Adida, 2008; Adida et al., 2009). In other words, *only 8 (22%) of all the voters who participated in this study completed a full verification* in which they checked their vote (associated with the smart ballot tracker that they recorded) on the online bulletin board. Voters who only partially verified did not do this—and instead relied on methods like viewing a vote confirmation email—likely because instructions on the vote-verification process were not provided to the users. Even though this deficiency did not lead to a total failure, the voters did not use the system as the designers intended in order to make sure that the system is working properly and that there is not any evidence of tampering.

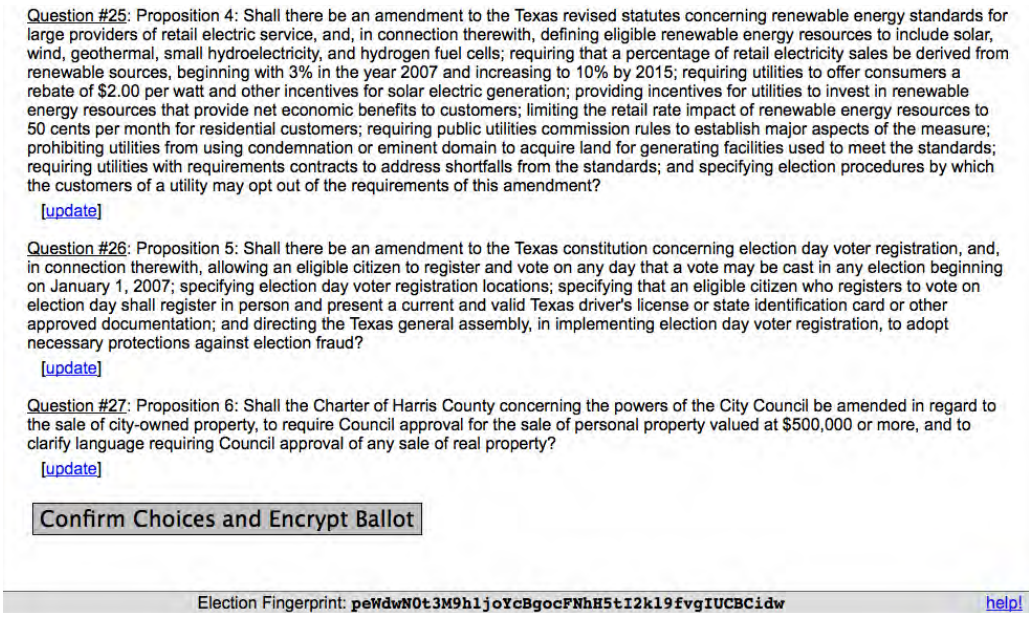


Figure 3. Helios vote selection review screen

Prêt à Voter

Vote Casting

As can be seen in Table 2, when voting with the PaV voting system, 8 (22%) of participants **[F]** did not scan in their ballot before placing it in the ballot box. It should be pointed out that the simple, single-use scanner (see Figure 4) was easily accessible. It automatically fed and scanned the ballot cards when they were placed into the feeder slot. In addition, the ballot box was placed off to the side of the vote-casting station. There are several design deficiencies that might have contributed to this scanning error resulting in a failure to vote cast.

[F1, K1] *No physical mechanisms are on the ballot box preventing an unscanned ballot from being placed in it.* The ballot box used in this study was not designed to keep unscanned ballot cards (i.e., uncast ballots) from being placed in it. A design solution might be to place the scanner behind the ballot box opening. A voter would simply drop in their ballot into the ballot box for it to be both scanned and stored (for an example see Belton, Kortum, & Acemyan, in press).

Table 2. Errors and Likely Contributing Design Deficiencies that Led to PaV Vote-Casting and Vote-Verification Failures

Number of Participants (Percentage of Participants)	**Error**	**Contributing Design Deficiencies**
Vote Casting		
8 (22%)	Did not scan-in ballot; placed it directly in the locked ballot box **[F]**	• No physical mechanisms are on ballot box preventing an unscanned ballot from being placed in it **[F1]** • System does not support prior voting mental model **[F2]** • No inline instructions **[F3]**
5 (14%)	Could not operate scanner **[G]**	• Scanner is not explicitly identified **[G1]** • Operating instructions are not available **[G2]** • Design of scanner **[G3]**
1 (3%)	Refused to detach the candidates list and then proceed with the voting process due to being confused **[H]**	• Numerous novel steps are required to cast a ballot **[H1]** • No inline instructions **[H2]**
Vote Verification		
14 (38%)	Did not cast a ballot **[I]**	• See above deficiencies
1 (3%)	Could not navigate to the verification website **[J]**	• Requires use of the internet **[J1]** • Website address is too complex **[J2]**

Note: This table includes observed user errors that led to a vote casting and/or vote verification failures. It does not include every critical incident observed throughout the study.

[F2, K2] *System does not support prior voting mental models.* The tested version of PaV does not support participants' general how-to-vote procedure for voting with a paper ballot (Acemyan, Kortum, Byrne, & Wallach, in press; Norman, 1983). Specifically, in elections that use a paper voting method, voters typically would complete a paper ballot and then place it into a locked ballot box to cast it. Therefore, in this study when participants used PaV to vote—which also requires the use of a paper ballot—they likely referenced their general model for voting with this type of ballot and then dropped it into the box instead of first scanning it. Skipping this critical step resulted in a failure to vote. To support these prior voting models formed through experience and general knowledge (Kaplan & Kaplan, 1983), the PaV voting system should mirror the typical how-to-vote-with-a-paper-ballot procedure as closely as possible.

Figure 4. Photograph of the scanner used in this research project for both PaV and Scantegrity II systems

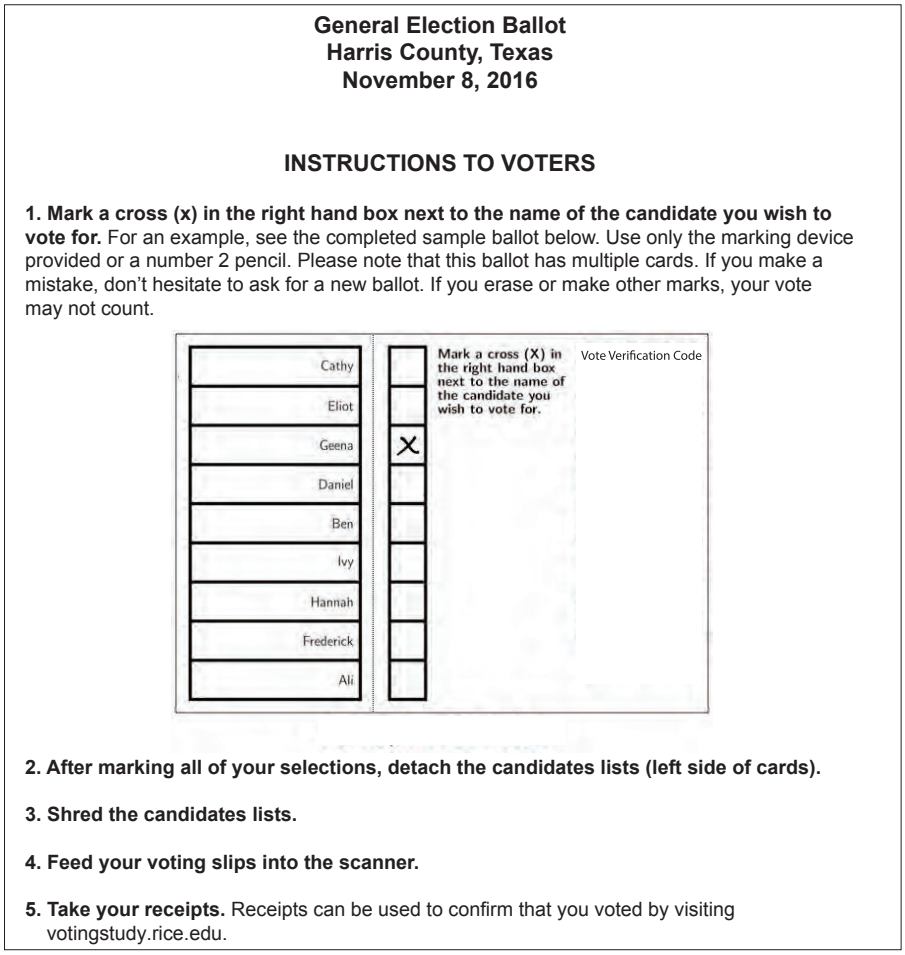
[F3, H2, K3, O3] *No inline instructions.* PaV does not provide the voters with instructions when they are needed at key points in the voting process. Instead voters are given lengthy instructions on a single page before they start the voting process (see Figure 5). This has the potential to pose a large cognitive load on the user (Johnson, 2010) who is voting with a novel system requiring them to do unusual things in the context of voting, like tearing their ballot in half. A system improvement would be to give the voter clear, concise instructions at each step in the vote-casting process—allowing voters to focus on their task while working within their short-term memory capacity (Shneiderman, 1987). Doing so might also prevent voters from accidently shredding their instruction sheet and then having no idea how to proceed, which happened to one participant.

Five more participants did not cast their ballots because they **[G]** could not identify and/or operate the scanner. Three potential design deficiencies have been identified.

[G1, L1] *Scanner was not explicitly identified.* Some voters had never used, or seen, a scanner before. This lack of previous exposure made it difficult to identify the single-function scanner (see Figure 4) and/or differentiate it from a single-function printer. Labeling the equipment with both the word "scanner" and a visual icon might allow some people to at least identify the piece of equipment. Of those that could identify the scanner, some participants expressed that they were afraid of it because they have never scanned a document before. Even though a scanner is a relatively common piece of equipment, it should not be assumed that every voter knows how to use even a simple one. Thus there should be support in identifying equipment and operating it with confidence.

[G2, L2] *Operating instructions for scanner are not available.* Inline instructions for operating the scanner were not provided to participants. Especially for participants who never used a scanner before, it would have been helpful to tell them how to operate the scanner, even at a global level: e.g., "Insert ballot cards in feeder slot for automatic scanning. No other action is required."

[G3, L3] *Design of scanner is not optimal*. In order for voters to not have to learn how to identify and operate a scanner to vote cast, the scanner could be redesigned. If the scanner is placed inside the ballot box so that voters only have to drop in the ballot (a step that they already familiar with) and the scanner—*hidden from view*—completes the process, then the voters would not have to worry about "scanning." (For a specific design suggestion, see Belton et al., in press.)



General Election Ballot
Harris County, Texas
November 8, 2016

INSTRUCTIONS TO VOTERS

1. Mark a cross (x) in the right hand box next to the name of the candidate you wish to vote for. For an example, see the completed sample ballot below. Use only the marking device provided or a number 2 pencil. Please note that this ballot has multiple cards. If you make a mistake, don't hesitate to ask for a new ballot. If you erase or make other marks, your vote may not count.

2. After marking all of your selections, detach the candidates lists (left side of cards).

3. Shred the candidates lists.

4. Feed your voting slips into the scanner.

5. Take your receipts. Receipts can be used to confirm that you voted by visiting votingstudy.rice.edu.

Figure 5. PaV instruction card, which was first page of ballot

[H, H1] *Numerous novel steps are required to cast a ballot.* One participant (3%) refused to detach their candidates list and then proceed with the voting process because they were confused. They did not understand why they would detach their selections from the people that they voted for and then shred them so there was not any record. Requiring the voter to perform so many atypical, unexpected steps seemed to erode their confidence. It was also clear that they did not understand why they were doing what they were doing to make their votes private (with the PaV system, a randomly ordered list of candidates is separated from the voter's selections and then shredded so that another person could not look at the ballot cards to figure out for whom they voted). To overcome this design deficiency in the future, the system should align as close as possible with typical how-to-vote procedures (Acemyan et al., in press; Nielsen & Molich, 1990), and if a voter must perform a novel procedure, it should be easy to accomplish. It would also help the user if they understood why they were performing the action.

Vote Verification

Fourteen (38%) participants did not verify that their vote was cast with PaV because they **[I]** did not cast a ballot. Addressing system design deficiencies would likely reduce this rate of failure. Of the participants who were able to cast a ballot (23 in total), one (4%) was not able to verify their vote. The one participant who was not able to check on their vote failed because they **[J]** could not navigate to the vote-verification website. This error can be accounted for by two likely design deficiencies:

[J1, J2] *Voter must use the internet and website address is too complex.* PaV's vote-verification system required the participants to use the internet. However, one participant could not use it. In particular, they did not know in which field they should enter the website address; they kept entering it into the search engine's search field, a common error among novice internet users (Sisson, 2014). Repeatedly, this participant also did not correctly type the address, "votingresearch.rice.edu"—highlighting that the verification sites should have the simplest and shortest address possible.

Scantegirty II

Vote Casting

As can be seen in Table 3, 16 (49%) participants were not able to cast their ballot with the Rice implementation of Scantegrity II.

Table 3. Errors and Likely Contributing Design Deficiencies that Led to Scantegrity II Vote-Casting and Vote-Verification Failures

Number of Participants (Percentage of Participants)	**Error**	**Contributing Design Deficiencies**
Vote Casting		
16 (43%)	Did not scan-in the ballot; placed it directly in the locked ballot box **[K]**	• No physical mechanisms are on the ballot box that prevented an unscanned ballot from being placed in it **[K1]** • System does not support prior voting mental models **[K2]** • No inline instructions **[K3]**
1 (3%)	Could not operate scanner **[L]**	• Scanner is not explicitly identified **[L1]** • Operating instructions are not available **[L2]** • Design of scanner **[L3]**
1 (3%)	No data available **[M]**	
Vote Verification		
18 (49%)	Did not cast a ballot **[N]**	• See above deficiencies **[N1]**
3 (8%)	Did not record ballot ID **[O]**	• No inline instructions **[O1]** • Ballot ID is not located in a prominent location on the ballot **[O2]** • Ballot ID is too long and complex, which deterred voter from making the effort to record it **[O3]**
1 (3%)	Could not navigate to the website **[P]**	• Voter did not know how to use the internet **[P1]** • Website address too complex **[P2]**

Notes: This table includes all observed critical incidents that led to vote casting and vote verification failures. The table does not include every critical incident observed throughout the entire study.

Four of the 37 participants (11%) did not use the special marking device as instructed, and instead used pen or pencil to complete their ballots. One of the Scantegrity II developers pointed out that completing a ballot with pen or pencil "does not necessarily invalidate the ballot" (A. Sherman, personal communication, August 20, 2014). However, three of these four participants never scanned in their ballot, meaning they never cast it. Therefore, the vote-casting rate would fall by 1 participant (3%) if failing to use the marking device invalidates the ballot. In this report of the data, a ballot was considered valid regardless of the marking device used—even though we explicitly asked participants to take all steps necessary to check on their votes after the election, which would include using the decoder pen to mark each selection.

[K, L] The same ballot box and scanner used in the PaV mock election were used with this system. For this reason, the other possible design deficiencies for vote casting are identical to those provided in the PaV section above.

Vote Verification

Eighteen (49%) participants were not able to verify their vote because they **[N]** did not cast a ballot. Of those that were able to cast their ballots, four (21%) were not able to verify successfully.

Three participants were not able to verify that their ballot was cast because they **[O]** did not record the ballot ID, which is required to log in to the verification website (see Figure 6 for an example of this error). The following are the associated design deficiencies:

[O1] *No inline instructions.* Like PaV, the tested version of Scantegrity II did not give voters simple, instructions at the time that they were needed during the vote-casting and vote-verification processes. Instead, long blocks of instructions were found at the tops of the ballot and vote confirmation sheets. For the full explanation of this design deficiency, refer to the PaV section.

[O2] *Ballot ID is not located in a prominent location on the ballot.* Per the City of Takoma Park, Maryland, Municipal Election, November 8, 2011 sample ballot, the ballot ID was placed on the front, lower right corner of this study's ballot. A similar font and font size was also used. It is possible that placing the ballot ID in a location that was not prominent, and in the voters' visual periphery (Johnson, 2010, p.65), might have contributed to voters not being able to easily detect it—especially since there was a large quantity of information displayed (Tullis, Tranquada, & Siegel, 2011) on the ballot (i.e., election title heading, voting instructions, races with candidates, revealed codes, and the unique ballot identifier).

[O3] *Ballot ID is too long and complex.* The ballot identification on every participant's ballot was "HC-2016-11-08-420795502." HC stood for Harris County, 2016-11-08 was the hypothetical date for the mock election, and 420795502 was the unique number associated with the particular ballot (this 8-digit number was based on a sample of e2e ballot identification codes, many of which were even longer). This study's ballot ID was not the same as the ID shown on the illustrative Scantegrity II ballots (i.e., #0001) featured in published and website materials (e.g., https://www.usenix.org/legacy/events/evt08/tech/full_papers/chaum/chaum_html/index.html). The published Scantagerity II illustrative ballot ID was too simple and not representative of a ballot ID used in an actual election.

Because the ID used in this study was so long and complex, it might have deterred participants from putting in the effort and time to record it. That is the perceived cost was too high for the return gain of being able to verify (Johnson, 2010), a goal secondary to vote casting. Or perhaps participants did not recognize the benefit of being able to verify, so they did not want to transcribe the ID. Based on the observed failures, if a ballot uses a code that voters must transcribe, those codes should be short, simple, and chunked (Miller, 1956) so that each group of numbers/letters is composed of 3-4 units—this will help reduce both voters' cognitive load and the opportunity for them to make mistakes.

One participant, **[P]** could not use the internet to get to the website, the same error and **[P1, P2]** contributing design deficiencies encountered by a PaV voter.

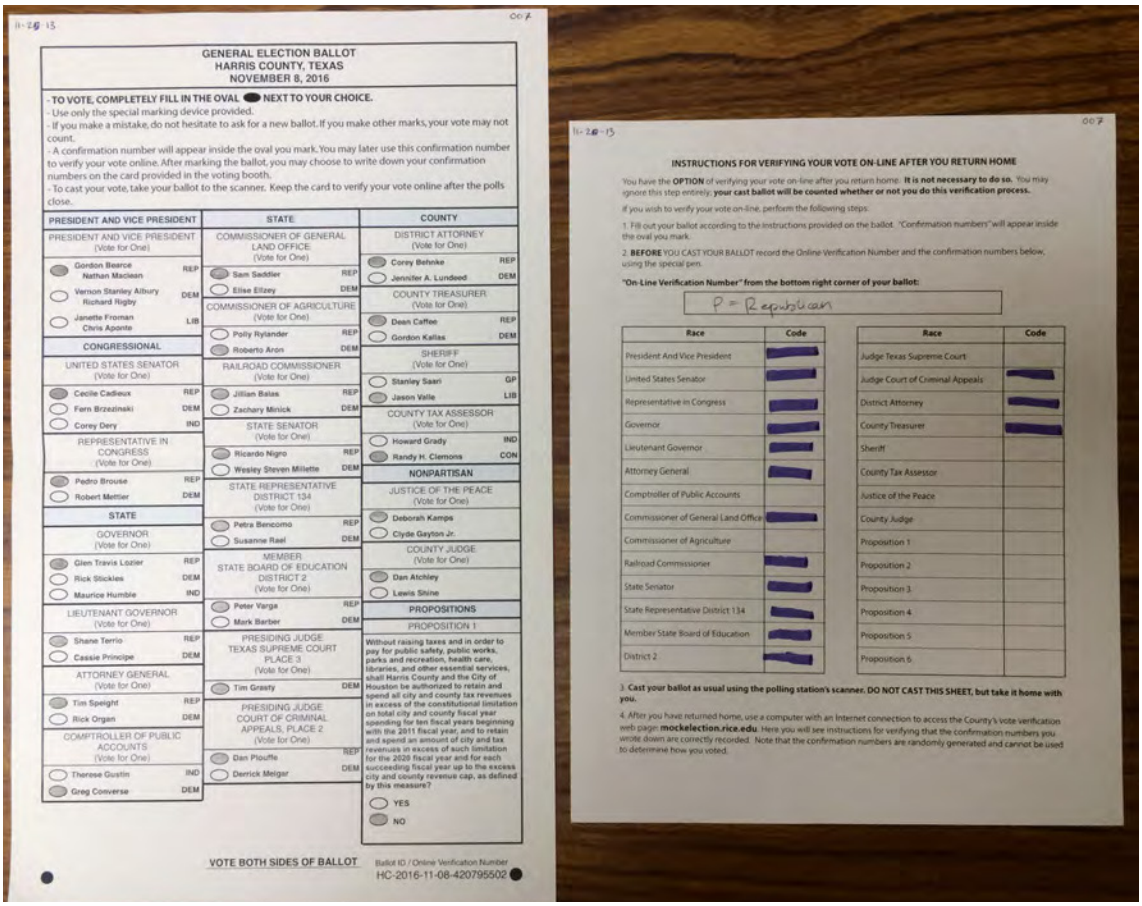


Figure 6. One participant's Scantegrity II ballot is shown left, and their vote confirmation sheet is shown right. For two reasons, this participant was not able to check on their votes. First, they did not use the decoder pen when completing their ballot, so the codes associated with each selection were not revealed or recorded. Second, they failed to record their ballot ID (located in the bottom right corner of their ballot) on their confirmation sheet (the top box is the area of the form reserved for the ballot ID). On account of these two errors, they did not record any information to track their ballot online.

Of the 15 (40%) of study participants who were able to complete some form of vote verification, only three (21%) of these completed verification procedures as outlined in the published papers describing Scantegrity II (e.g., Carback et al., 2010; Chaum et al., 2008). All told, *only 8% of all the voters who participated in the mock election completed a full verification* in which they correctly recorded their ballot ID, correctly wrote down all 27 codes, and looked at the vote-verification webpage associated with their ballot. Since there is no data to confirm that participants actually compared all 27 transcribed codes to those displayed on the site, the full verification rate has the potential to be even lower than 8%. Voters likely only partially verified because too much of a mental workload and physical burden was placed on them. After completing a ballot, participants had to transcribe by hand a long, complex ballot ID and small codes revealed inside each selected bubble. Then the subjects had to navigate to the vote-verification website, and only if they wrote down all of the information correctly could they type in the case sensitive ballot ID and compare all the codes one by one.

Another major problem with the Scantegrity II verification system was that there were too many opportunities for voters to make errors while transcribing and then comparing the verification information. Only 62% of all participants wrote down the ballot ID correctly, and just 8% recorded all 27 codes accurately. The ballot ID and each code that had to be written down on the confirmation sheet is an opportunity for the voter to make a mistake. Then there are even more opportunities for errors when the voters type the ballot ID into the verification system and

compare their recorded information with the information the system recorded. What if they notice one of the codes that they recorded does not match the corresponding code the system displays on the vote-verification screen? At this point the voter can contest the election. This brings up even more issues—such as the lack of concrete, specified procedures from the voters perspective to handle this type of event (e.g., who do they contact, how do they contact them, and what information do they need to provide?). This is applicable also to Helios and PaV. In addition, another concern about the Scantegrity II system is the possibility of potential deliberate attacks on the system. For example, imagine a voter claims the systems' online codes do not match their written ones, when in fact the malicious voter intentionally wrote-down the wrong codes so that they could report that the system was not operating as expected. This type of event could end up costing a precinct resources.

For all of these reasons discussed in this section, Scantegrity II had one of the weakest vote-verification systems tested in this study. Even though the identified system design deficiencies did not lead to a total failure, it is still highly problematic because the voters did not use the system as the designers intended in order to check that their votes were recorded accurately by the system.

DISCUSSION

This study, which focused on all of the observed errors resulting in a total failure to vote cast and vote verify, found these errors were caused by only a few contributing design deficiencies. This is positive because a small number of fixes to the system designs could likely solve the majority of the issues that caused voters to fail at vote casting. As for the verification methods, if the voters need to complete full vote verifications for security reasons, then the Helios and Scantegrity II methods should be heavily reconsidered to reduce the amount of physical and cognitive work placed on voters.

Some Cautionary Notes

Errors and contributing design deficiencies listed for each system are not punch lists.

While every usability issue observed while conducting the study was not identified in this paper, the data do provide direction for improving the systems to increase the likelihood that voters will fulfill the objectives of vote casting and vote verifying. What this paper does not provide is a punch list of elements to fix. First, even if every possible system design deficiency identified in this paper is addressed, the systems still might not be highly usable. The identified usability issues might have masked problems that have yet to be identified. Hence, once the outlined problems are fixed, new problems might be found that can only then be recognized. Second, there is the possibility that the design deficiencies presented in this paper may not be the root cause of the failures observed. Even though the deficiencies are based on basic human factors and human-computer interaction principles, until additional data is collected, there is no way to know for certain that they account for the observed voter behaviors. Third, a voting research team might think that they have the perfect solution for a problem, when in fact the solution does not fully resolve the problem, and may even causes new issues to arise. For instance, Scantegrity II, as deployed in the 2011 Takoma Park City Municipal Election, had the scanner attached to the ballot box (scantegrity.org). This design (which differs from published papers about the Scantegirty II system, e.g., Chaum et al., 2008; Carback et al., 2009; Chaum et al., 2009; Sherman et al., 2009) prevented the problem of failing to scan the ballot before placing it into the ballot box. What is unknown is if the voters had / would have trouble using the new scanner setup, or voters who have never scanned documents before were / would be still afraid to use it. In that case the design solution would still be insufficient and work would need to be done to improve system usability. For these reasons, the errors and contributing design deficiencies outlined here should be considered when designing any voting system so that the same mistakes are not repeated. With respect to the three systems tested in this paper, further testing must take place after making design modifications to assess if the system is more usable and all issues have been addressed.

Helping voters, instructions, and training are not "fixes."

It might be argued that if these systems had been used in an actual election, some of the behavioral errors and consequent system failures could have been avoided if the voters had asked an election official for assistance. This may be true. Assistance, offered at the right time, can prevent usability errors or rectify those that have occurred (similarly, assisting participants in a usability study greatly impacts test results; Rubin and Chisnell, 2008). While there are always officials present at polling sites, they should not be the first line of defense, and voters should not have to rely on them to successfully use the systems. There are high costs associated with voters asking someone to help them: voters might become frustrated and give up voting before asking for help, they might avoid voting in the future, their votes could be revealed to another person, officials might have to be trained to know how to address

every possible usability issue, election officials might not be able to assist in a timely manner in busy polling stations, the availability of help is not uniform across jurisdictions as some locations do not have a plentiful supply of well-trained poll workers, racial bias in poll workers might result in race-based discrimination when helping voters complete and cast ballots (Page & Pitts, 2009)—meaning there is the potential for some races to be helped more effectively than other races, and there is the potential for voting times to increase, making lines at some polling stations even longer.

It is also not wise to rely heavily on instructions and/or training in order for people to be able to use a system with usability problems, even if it is new and different—as is the case with all three of the e2e systems studied in this research. Instead it would be best to redesign the systems in order to make them easier to use and minimally reliant on instructions.

When Scantegrity II was used in a real election, "instructions along waiting lines, instructional video[s]… instructions in marking booth[s], instructions in newspapers and TV prior to election, [and] verbal instructions as people entered" were relied upon in addition to the instructions printed on the ballot and confirmation sheet (A. Sherman, personal communication, August 20, 2014). The researchers were trying to help the voters understand what to do and convey that they were able to verify their votes afterwards if they completed extra steps while voting. Despite these efforts, some voters ignored all instructions and ended up "not realiz[ing] that they could verify their vote and that they had to make a receipt to do so" (A. Sherman). It is also possible that not every voter who participated in the election received the training or encountered all instructions sets. So rather than trying to change user behavior, instead change the system design (Ross, 2010) to make sure the system is usable in the first place. If a system is highly usable, then there is no need for training—saving everyone time and money—and figuring out how to use the system becomes more effective (Rohn, 2005). Further, training and instructions do not fix inefficiencies of use (i.e., every user still has to go through inefficient steps) or improve user satisfaction (i.e., being able to accomplish a goal does not necessarily mean it was a good experience; Ross, 2010), two other aspects of usability not expressly addressed in this paper's formative analysis.

In summary, instructions and videos do not fix a system's usability problems and should not be used as a first line of defense, especially in a walk-up-and-use system. The systems must be designed so that any user can walk up and effortlessly use them, without any assistance, which is especially important in voting. Simple, inline instructions written in plain language might even help to achieve this goal (for more information about writing instructions, refer to Dana Chisnell's *Writing Instructions Voters Understand*, 2013).

Test, iterate, and retest.
Summative testing is a great tool to understand the current state of e2e system usability. Formative studies are useful for identifying particular designs that need improved upon or avoided altogether moving forward. It is expected that using both of these methods early in the development process would reveal many usability deficiencies. It is unrealistic to expect a perfectly secure and usable system in a single pass of development because addressing both system security and usability at the same time is challenging, yet possible (Goodman et al., 2012). This is why system developers need to continually refine their voting systems and test them with real users. This process will then arm them with empirical data—versus anecdotes and personal opinions—to support a system design, modification, or equipment setup as being an usability improvement (or detriment) for voters.

Future Research
Besides conducting summative and formative usability studies on the next generation of e2e voting systems, future research must address the usability of e2e voting systems for those with reading, cognitive, and physical disabilities. For example, after observing this study's participants, we believe that Helios would be the most usable for voters with several classes of disabilities such as movement and vision. Helios is implemented in an electronic format, meaning existing software and hardware that is used by the disabled to interact with computers and DREs can be paired with the voting system. Non-sighted voters could use screen readers, and voters with movement impairments could use equipment like jelly switches and sip-and-puff devices to help them vote.

In contrast, PaV and Scantegrity II would likely pose many extremely difficult accessibility problems. For example, how would non-sighted voters complete their paper ballots? Even if a device was available to read the PaV ballot to them, these voters would have difficulty knowing which pieces of paper to shred (i.e., the candidates lists) and which pieces to eventually scan (i.e., the completed ballot cards). The receipt issued to the non-sighted voters would

likely be meaningless because visual information is displayed on a ballot card. It is unclear if even a reading device would be able to help them interpret it. Similar challenges are associated with Scantegrity II, as non-sighted voters would not be able to read the revealed codes inside each selected bubble and would probably struggle to create their own receipt without assistance. While Audiotegrity has been proposed as a solution for unsighted voters to complete their Scantegrity ballot and print-off a confirmation card that lists the ballot ID and confirmation numbers for each of their selections (Kaczmarek et al., 2013), testing would still need to be conducted to assess the usability of the system, especially since Audiotegrity does not solve problems like how an unsighted voter would check that their printed ballot is correctly marked and that the confirmation card lists the correct confirmation numbers.

Voters with other physical disabilities, like movement impairments, will also probably have trouble with some of the tasks required to vote. For instance, consider the process of voting with PaV. It requires a voter to tear the ballot in half, shred the candidates list, and feed the remaining ballot cards into a scanner. In this study, a participant was observed who could not use the right half of their body. They resorted to using their teeth to complete these tasks.

Another participant in this study had problems standing and moved slowly. While vote casting with Scantegrity II, they filled in the bubble for their selection very slowly, shuffled back and forth between the ballot and vote confirmation sheets, switched between writing devices (a decoder pen was used on the ballot and a pen was used to record the codes on the vote confirmation sheet), and carefully wrote down the code that was revealed before going onto the next race. Since the task was so labor and time intensive for the participant, they kept sitting down to rest. This same participant seemed to more easily use the Helios and PaV systems since they only required them to interact with a single interface while completing their ballot. For reasons like these, research must be conducted to identify usability issues that the disabled will face and determine if and how the issues could be solved.

CONCLUSION

This paper is not intended to be a directed criticism of any of the tested e2e systems that we selected. Rather, the intent of the paper was to demonstrate that deviations from the standard paper/ballot box voting method can lead to failures in the ability of the user to successfully complete their desired task. Going forward, we need to be certain to understand any changes that are made from the voters' expected voting model and ensure that these deviations have minor repercussions for user success. Accordingly, conducting controlled, high-quality usability evaluations of these new systems is imperative.

In summary, the complex security mechanisms found in e2e voting systems likely do not comprise an insurmountable barrier to the development of usable designs. Even though it is difficult to create error-resistant systems for diverse populations, it is possible. By addressing the design deficiencies and implementing solutions using industry-standard methods like iterative design, secure e2e systems can become usable enough to be deployed in large-scale, actual elections.

ACKNOLEDGEMENTS

This research was supported in part by the National Institute of Standards and Technology under grant #60NANB12D249. The views and conclusions expressed are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of NIST, the U.S. government, or any other organization.

REFERENCES

Acemyan, C.Z., Kortum, P., Byrne, M.D., & Wallach, D.S. (2014). Usability of voter verifiable, end-to-end voting systems: Baseline data for Helios, Prêt à Voter, and Scantegrity II. *USENIX Journal of Election Technology and Systems (JETS), 2*(3), 26-56.

Acemyan, C.Z., Kortum, P., Byrne, M.D., & Wallach, D.S. (in press). Users' mental models for three end-to-end voting systems: Helios, Prêt à Voter, and Scantegrity II**.** *Lecture Notes in Computer Science.*

Adida, B. (2008). Helios: Web-based open-audit voting. *Proceedings of the 17th USENIX Security Symposium*, *USA, 17*, 335-348.

Adida, B., De Marneffe, O., Pereira, O., & Quisquater, J. J. (2009). Electing a university president using open-audit voting: Analysis of real-world use of Helios. *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections, USA, 18*.

Belton, G., Kortum, P., & Acemyan, C.Z. How hard can it be to place a ballot into a ballot box? Usability of ballot boxes in tamper resistant voting systems. *Journal of Usability Studies*, in press.

Carback, R., Chaum, D., Clark, J., Conway, J., Essex, A., Herrnson, P.S., ... Vora, P.L. (2010). Scantegrity II municipal election at Takoma Park: The first e2e binding governmental election with ballot privacy. *Proceedings of the 19th USENIX Security Symposium*, *USA, 19*.

Chaum, D., Carback, R.T., Clark, J., Essex, A., Popoveniuc, S., Rivest, R.L., … A.T., Vora, P.L. (2009). Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Information Forensics and Security, 4*(4), 611-626.

Chaum, D., Essex, A., Carback, R., Clark, J., Popoveniuc, S., Sherman, A., & Vora, P. (2008). Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy, 6*(3), 40-46.

Chisnell, Dana. (2013). *Writing instructions voters understand* (Field Guides to Ensuring Voter Intent, Vol. 02). Retrieved from the Presidential Commission on Election Administration website: https://www.supportthevoter.gov/files/2013/08/Field-Guide-Vol-02-20130620.pdf

Everett, S. P. (2007). *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection* (Doctoral dissertation, Rice University). Retrieved from http://chil.rice.edu/alumni/petersos/EverettDissertation.pdf

Del Galdo, E.M., Williges, R.C., Williges, B.H., & Wixon, D.R. (1986). An evaluation of critical incidents for software documentation design. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting, USA, 30*(1), 19-23.

Goodman, E., Kuniavsky, M., & Moed, A. (2012). Balancing needs through iterative development. In *Observing the User Experience: A Practitioner's Guide to User Research* (pp. 21-44). Waltham, MA: Morgan Kaufmann.

Green, R. (1990). Human error on the flight deck. In D.E. Broadbent, A. Baddeley, and J.T. Reason (Eds.), *Human Factors in Hazardous Situations* (pp. 503-512). Oxford: Clarendon Press.

How to Vote with Scantegrity. (n.d.) Retrieved from scantegrity.org

Itoh, K., Omata, N., & Andersen, H.B. (2010). A human error taxonomy for analyzing healthcare incident reports: Assessing reporting culture and its effects on safety performance. *Journal of Risk Research, 12*(3-4), 485-511.

Johnson, J. (2010). *Designing with the mind in mind: Simple guide to understanding user interface design rules.* Burlington, MA, Morgan Kaufmann.

Kaczmarek, T., Wittrock, J., Carback, R., Florescu, A., Rubio, J., Runyan, N., Vora, P.G., & Zagorski, F. (2013). Dispute resolution in accessible voting systems: The design and use of Audiotegrity. *Lecture Notes in Computer Science, 7985*, 127-141.

Lewis, J.R. (1994). Sample sizes for usability studies: Additional considerations. *Human Factors: The Journal of the Human Factors and Ergonomics Society, 36*(2), 368-378.

Lundin, D., & Ryan, P.Y. (2008). Human readable paper verification of Prêt à Voter. In S. Jajodia & J. Lopez (Eds.), *Computer Security – ESORICS 2008: Proceedings of the 13th European Symposium on Research in Computer Security, Malaga, Spain, October 6-8, 2008* (pp. 379-395). Berlin, Germany: Springer Berlin Heidelberg.

Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review, 63*, 81-97.

Nielsen, J. (1997). *How Users Read on the Web.* Retrieved from http://www.nngroup.com/articles/how-users-read-on-the-web/

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. *Proceedings of ACM CHI'90 Conference*, *USA*, 249-256.

Norman, D.A. (1983). Some observations on mental models. In D. Gentner, & A.L. Stevens (Eds.). *Mental Models* (pp. 7-14). New York, NY: Lawrence Erlbaum Associates.

Page, A., & Pitts, M.J. (2009). Poll workers, election administration, and the problem of implicit bias. *Michigan Journal of Race & Law, 15*(1), 1-57.

Prêt à Voter. (n.d.). Retrieved from http://www.pretavoter.com

Reason, J. (1990). *Human error*. Cambridge, England: Cambridge University Press.

Rubin, J., & Chisnell, D. (2008). Conduct the test sessions. In *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests* (pp. 201-220). Indianapolis, IN: Wiley.

Rohn, J.A. (2005). Cost-justifying usability in vendor companies. In R.G. Bias, & D.J. Mayhew (Eds.), *Cost-Justifying Usability* (pp. 185-191). San Francisco, CA: Morgan Kaufmann.

Ross, J. (2010). *It's Not a Training Issue*. Retrieved from http://www.uxmatters.com/mt/archives/2010/12/its-not-a-training-issue.php

Ryan, P. Y., Bismark, D., Heather, J., Schneider, S., & Xia, Z. (2009). Prêt à Voter: A voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security, 4*(4), 662-673.

Ryan, P.Y., & Peacock, T. (2010). A threat analysis of Prêt à Voter. In D. Chaum, M. Jakobsson, R.L. Rivest, P.Y. Ryan, J. Benaloh, & M. Kutylowski, (Eds.), *Lecture notes in computer science: Vol. 6000. Towards trustworthy elections: New directions in electronic voting* (pp. 200-215). New York, NY: Springer.

Ryan, P.Y., & Schneider, S.A. (2006). Prêt à Voter with re-encryption mixes. In D. Gollmann, J. Meier, & A. Sabelfeld (Eds.), *Computer security – ESORICS 2006: Proceedings of the 11th European symposium on research in computer security, Hamburg, Germany, September 18-20, 2006* (pp. 313-326). Berlin, Germany: Springer Berlin Heidelberg.

Sanders, M.S., & McCormick, E.J. (1993). *Human factors in engineering and design*. New York, NY: McGraw-Hill.

Sauro, S. (2010). *Are The Terms Formative And Summative Helpful Or Harmful?* Retrieved from https://www.measuringu.com/blog/formative-summative.php

Schmettow, M. (2012). Sample size in usability studies. *Communications of the ACM, 55*(4), 64-70.

Sherman, A.T., Carback, R., Chaum, D., Clark, J., Essex, A. …. Vora, P. (2010). Scantegrity mock election at Takoma Park. *Electronic* Voting, 45-61.

Shneiderman, B. (1987). *Designing the user interface: Strategies for effective human-computer interaction.* Reading, MA: Addison-Wesley.

Sisson, D. (n.d.). *Assumptions about user search behavior.* Retrieved from philosophe.com/search_topics/user_behavior/

Spool, J.M. (1999). *Web site usability: A designer's guide.* San Diego, CA: Academic Press.

Staggers, N., & Norcio, A.F. (1993). Mental models: Concepts for human-computer interaction research. *International Journal of Man-Machine Studies, 38*, 587-605.

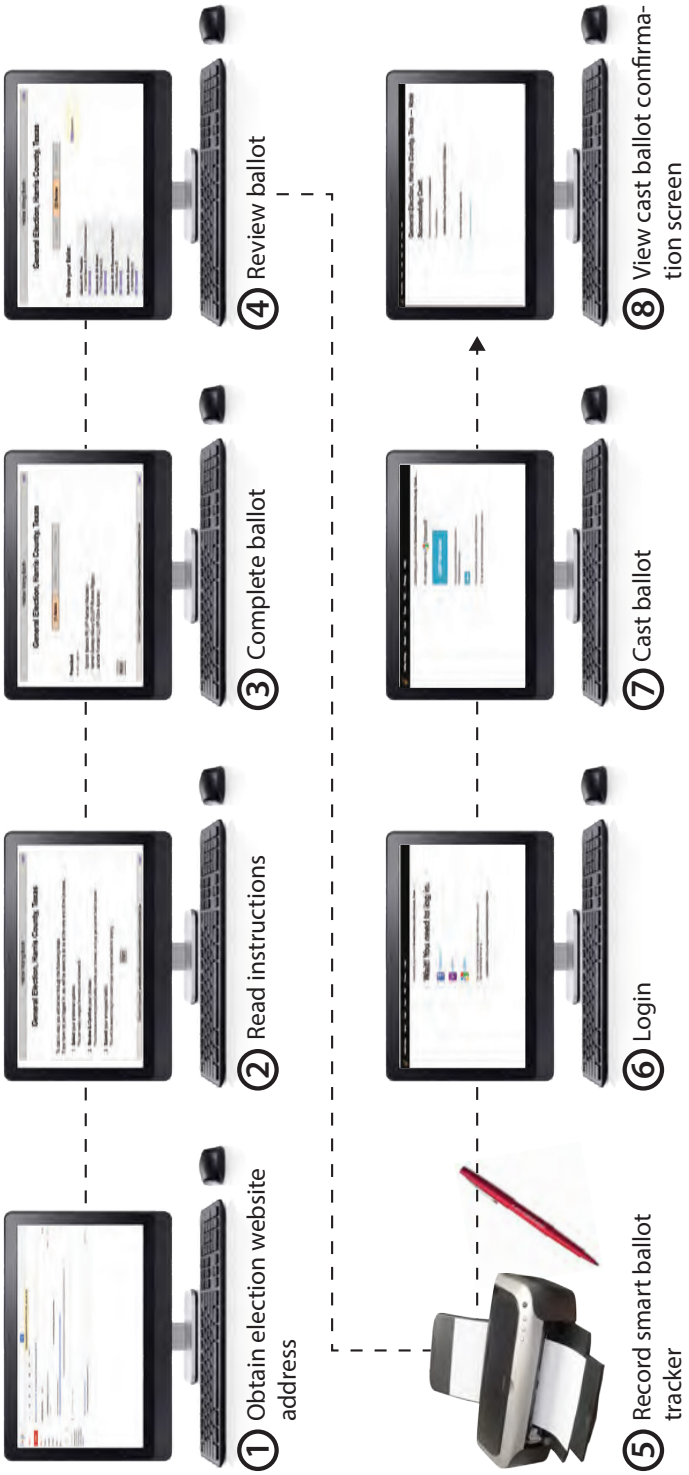
The City of Takoma Park, Maryland. (n.d.). Retrieved from http://www.takomaparkmd.gov

Tidwell, J. (2011). *Designing interfaces.* Sebastopol, CA: O'Reilly Media, Inc.

Tullis, T.S., Tranquada, F.J., & Siegel, M.J. (2011). Presentation of information. In K.L. Vu, & R.W. Proctor (Eds.), *Handbook of Human Factors in Web Design* (pp. 154-184). Boca Raton, FL: CRC Press.

APPENDIX A

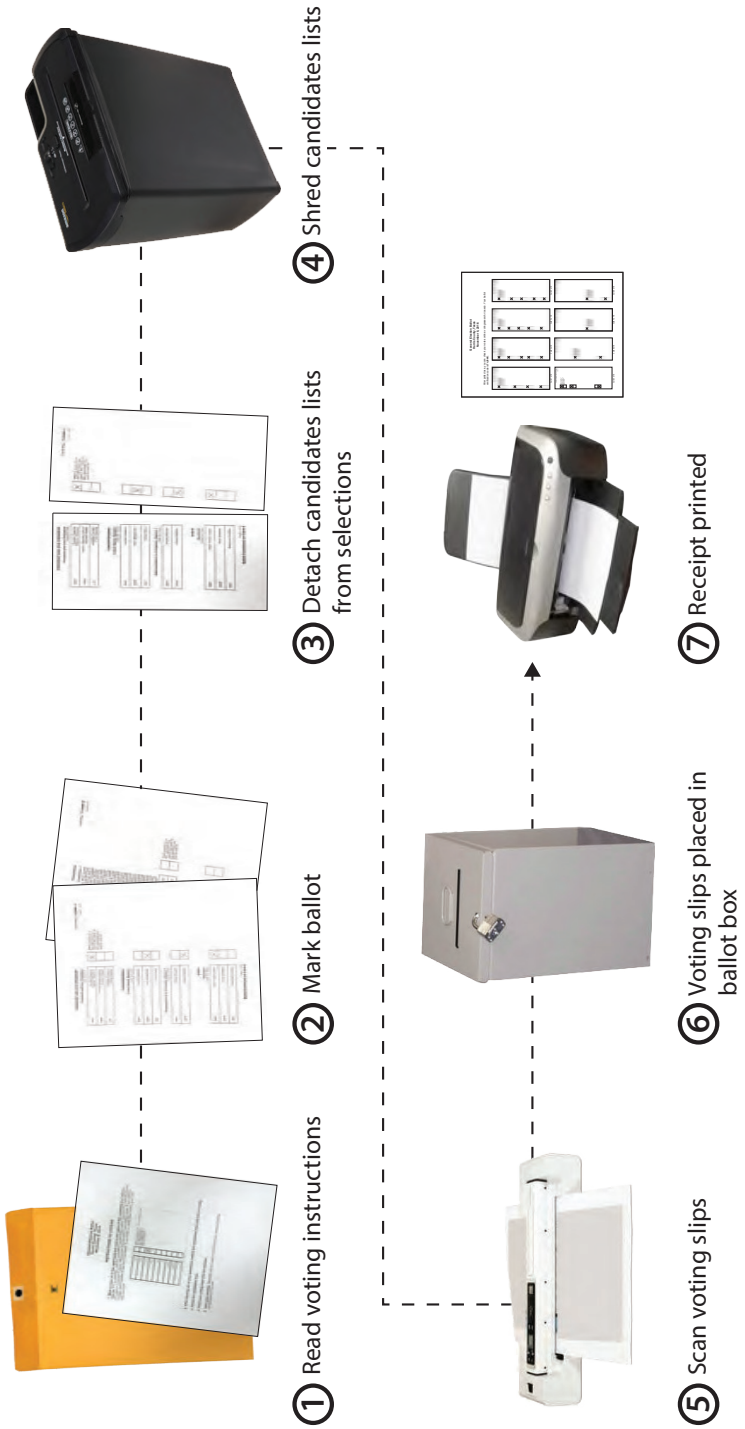
Typical Helios Vote-Casting Procedure



Typical Helios Vote-Verification Procedure

APPENDIX B

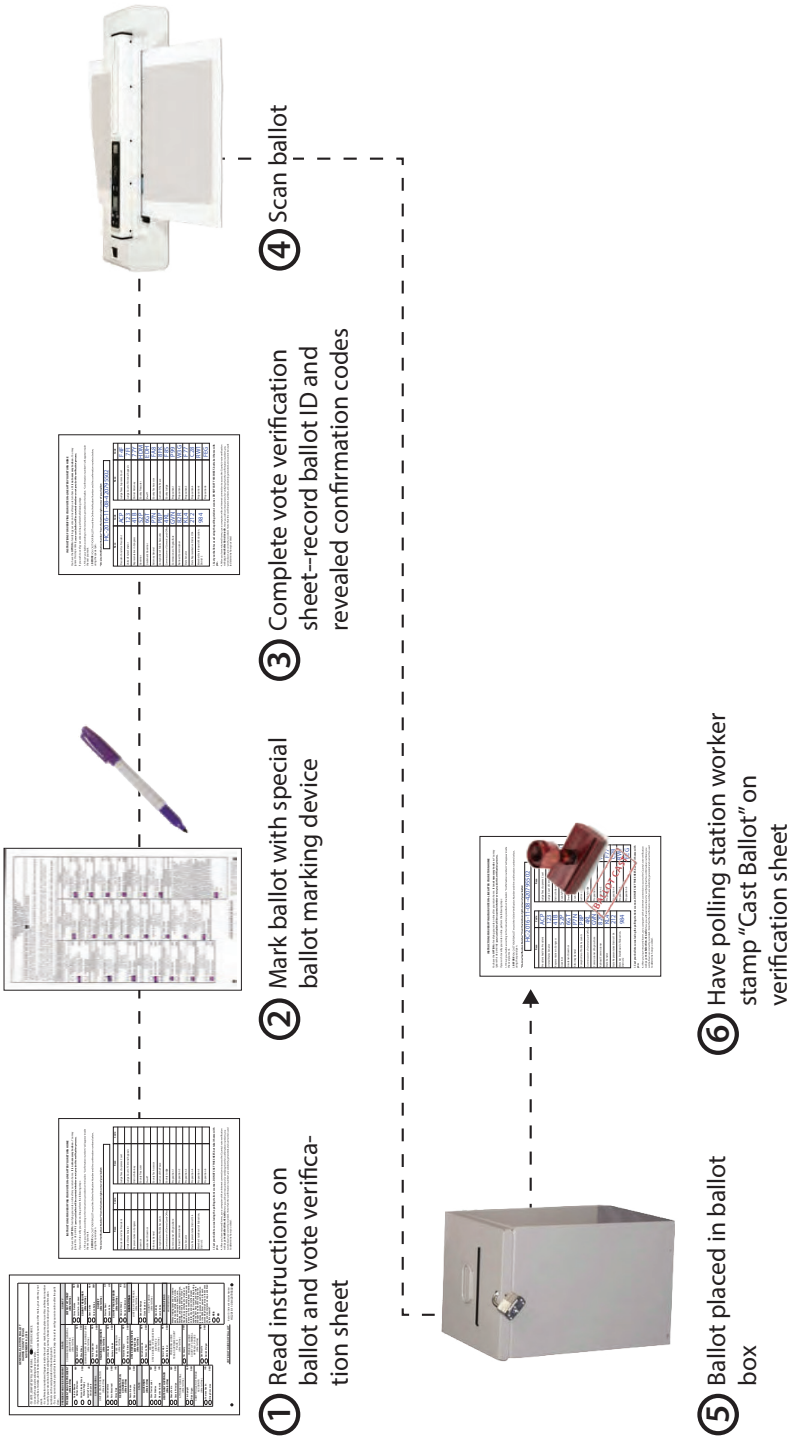
Typical Prêt à Voter Vote-Casting Procedure



Typical Prêt à Voter Vote-Verification Procedure

APPENDIX C

Typical Scantegrity II Vote-Casting Procedure



Note: Step six was based on *Scantegrity II: End-to-end Verifiability for Optical Scan Election Systems Using Invisible Ink Confirmation Codes* (Chaum et. al., 2008).

Typical Scantegrity II Vote-Verification Procedure

USENIX Journal of Election Technology and Systems (JETS)
Volume 3, Number 2 • August 2015
www.usenix.org/jets/issues/0302

Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting

Dirk Achenbach, Karlsruhe Institute of Technology
Carmen Kempka, NTT Secure Platform Laboratories
Bernhard Löwe, Karlsruhe Institute of Technology
Jörn Müller-Quade, Karlsruhe Institute of Technology

In a democracy, it is essential that voters cast their votes independently and freely, without any improper influence. Particularly, mechanisms must be put into place that prevent—or at least severely impede—the coercion of voters. One possible countermeasure to coercion is *revoting*: after casting a vote under coercion, the voter can re-cast and overwrite her choice. However, revoting is only meaningful as a strategy to evade coercion if the adversary cannot infer whether the voter has modified her choice—revoting needs to be *deniable*, while still being publicly verifiable. We define the notions of correctness, verifiability, and deniability for a tallying protocol which allows for revoting. We also present a protocol realizing these notions. To the best of our knowledge, our solution is the first to achieve both deniability and public verifiability without asking information about the voter's previously-cast ballots for revoting. A seemingly competitive line of work, started by the well-known work of Juels, Catalano, and Jakobsson, uses fake credentials as a strategy to evade coercion: the voter presents to the adversary a fake secret for voting. In this work, we extend Juels et al.'s work to achieve deniable revoting. Their solution also allows for revoting, however not deniably. Our solution supports fake credentials as an opt-in property, providing the advantages of both worlds.

1. INTRODUCTION

Free elections are the backbone of a democracy. In traditional elections, voters fill out their ballot in a voting booth to ensure the privacy of the vote. Because traditional presence elections are tedious, there is a significant interest in carrying out elections over the Internet. It seems impossible to give the same privacy guarantee that a voting booth does in an online setting. Further, advances in consumer electronics even call the privacy of the voting booth into question. To sell their vote, voters can easily record their choice with their smartphone [Post 2010; Benaloh 2013].

To mitigate these problems, several solutions have been developed to make elections resistant against such attacks. *Fake credentials* [Juels et al. 2005] and *revoting* [Post 2010; Volkamer and Grimm 2006] are two of these. While *fake credentials* successfully disable the adversary from obtaining a voter's credential by coercion, they are contradictory to a meaningful response whether the ballot has been cast successfully. *Revoting*, on the other hand, allows for a meaningful response.

The idea behind revoting is that, if a voter is allowed to overwrite her vote arbitrarily many times, she effectively cannot be coerced. We stress, however, that this proposed revoting must be perfectly deniable—no party, including all servers, must be able to tell whether a coerced voter has evaded coercion by revoting.

As a side-effect, deniable revoting is applicable to elections which allow the voter to change her mind after casting a legit vote. An example for this is delegated voting [Green-Armytage 2014; Miller 1969], in which the voter can delegate her vote to a so-called proxy, who then votes in her stead. She can later overwrite this delegation with a new delegation or by casting a vote by herself.

We investigate deniable revoting as a strategy for evading coercion. We build upon the well-known work of Juels, Catalano, and Jakobsson [Juels et al. 2010]. Specifically, we adapt their definitions of correctness and coercion-resistance to include revoting and its deniability. While their construction enables voters to re-cast their vote, their security model does not account for revoting as a strategy to evade coercion. Indeed, their construction allows the adversary to observe whether a ballot is superseded. We present a tallying protocol which allows voters to deniably re-cast their votes, thus making revoting a viable strategy to evade coercion. The macrostructure of our protocol is the same as that of Juels et al., hence it is compatible with fake voter credentials. To facilitate a simpler exposition, we do not investigate the possibility of offering both evasion strategies simultaneously.

Existing protocols that allow revoting either do not achieve both deniability and public verifiability at the same time, or require the voter to remember some information about previous votes [Kutyłowski and Zagórski 2007]. To the best of our knowledge, we are the first to propose a publicly verifiable method for enabling deniable revoting without requiring the voter to save state between votes.

There is one caveat to the revoting approach. If a coercer observes the voter's behavior until after the polls close, the voter cannot re-cast her vote and thus revoting does not work as a strategy for evading coercion. This caveat applies to the strategy in general and thus cannot be remedied by any protocol. For this reason, our adversarial model does not allow the adversary to observe the voter's behavior indefinitely—he must give up his control before the polls close. On the other hand, a coerced voter does not employ any evasion strategy during adversarial observation—she submits to the adversary's instructions completely.

1.1. Our Contribution

We investigate revoting as a strategy for evading coercion. To this end, we extend the well-established notions of correctness, verifiability, and coercion-resistance by Juels, Catalano, and Jakobsson [Juels et al. 2010] to take deniable revoting into account. Juels et al.'s approach also allows voters to re-cast their vote, but not as a strategy for evading coercion. In fact, an adversary can learn whether a specific ballot cast under coercion is superseded by a more recent one. Our work improves upon the method of Juels et al. of eliminating duplicates such that no adversary can tell how often (and when) a particular voter casts her vote. We present a method for counting votes which ensures the deniability of revotes, and prove the correctness of our protocol as well as the deniability of revoting. This work seeks to serve as a proof of concept and does not claim to provide an efficient solution.

1.1.1. Juels, Catalano, and Jakobsson's Approach. Let us briefly recap the approach of Juels, Catalano, and Jakobsson. To cast a vote, voters post their ballot on a public bulletin board, together with proofs of knowledge of their credential and the validity of their choice. After the voting phase, the tallying authority computes the tally in five steps:

(1) Check proofs: The tallying authority checks all proofs associated with the ballots and discards ballots with invalid proofs.
(2) Remove duplicate ballots: At most one ballot per credential is kept. To identify duplicate ballots, *plaintext equivalency test*s (PETs) are employed on the encrypted voter credentials. For a pair of ballots with identical voter credentials, a pre-determined policy decides which ballot is kept.
(3) Shuffle: The remaining ballots are shuffled.
(4) Check credentials: The tallying authority discards ballots with invalid voter credentials.
(5) Tally: All remaining valid ballots are decrypted and counted.

This solution uses fake credentials as a strategy to evade coercion. Revoting is supported in the sense that double votes are handled. However, voters cannot plausibly deny having revoted—this is not the aim of the construction. Imagine an adversary forcing a voter to cast a certain ballot in his presence. Since the ballots are only shuffled (Step (3)) after duplicates have been removed (Step (2)), the adversary can easily monitor if his observed ballot is deleted. This way, he can deduce that the coerced voter has later cast a ballot with the same credential. This does not impose a problem in Juels et al.'s approach—the observed ballot was cast with a fake credential, and does not need to be superseded to evade coercion. To employ revoting as a strategy for evading coercion however, the sorting process must ensure its deniability. A first step is to shuffle the list of ballots before removing duplicates. This conceals the chronological order of ballots, however. For two ballots cast with the same credential, it cannot be determined anymore which one was cast more recently. We devised a method of an "encrypted labeling" which allows voters to privately re-cast their ballot, while preserving enough information about the chronological order of ballots.

1.2. Achieving Deniable Revoting

In this section we sketch the challenges of deniable revoting, and how to overcome them, in more detail. For revoting to be suitable as a strategy for evading coercion, the voter must be able to *deniably* do so. Intuitively, we say a voter can deniably re-cast her vote if no adversary can, by his observations, tell whether she did. At the same time, we still require the tally to be computed correctly. In particular, at any time during the voting period, for each participating voter who has already cast a vote, there

is exactly one *valid* ballot (i.e. one ballot which counts as long as it is not superseded), so a re-cast ballot must invalidate this one ballot with matching voter credentials, and become the new valid ballot corresponding to this credential. These requirements impose several privacy challenges on the tallying process.

More concretely, the deniability requirement of the revoting process implies that the adversary must not be able to figure out how often a certain voter has re-cast her ballot. Further, he must not even infer which of the voters did revote at all. Also, if he can tell whether any particular ballot is part of the final tally, revoting is not deniable. Nevertheless, to maintain the universal verifiability of the tally, the correctness of each tallying step must be transparent. To give an intuition for the subtle challenges of authenticating re-castable ballots, we describe two exemplary attacks.

For the first example recap the attack described in the former section. There, the adversary is forcing the voter to cast a ballot in his presence. He can then observe the ballot appearing on the bulleting board right after. If this particular ballot is deleted or marked as invalid in the tallying process, he can deduce that his coercion attempt was not successful. As mentioned above, we employ encrypted labelling and re-encryption mixes after the casting phase to solve this problem.

While Step (2) in Juels et al.'s construction (see Section 1.1.1) does hide the identity of voters with duplicate ballots, it leaks information about the number of removed duplicates per credential. Consider the following attack, which is similar to the "1009 attack" described by Warren Smith [Smith 2005]: The adversary forces the voter to vote, in his presence, a number of times no other voter is likely to vote, e.g. exactly 1009 times. During the tallying phase he then verifies that the tallying authority indeed identifies 1009 duplicate ballots for some credential. This becomes fatal in the absence of fake credentials[1]: the adversary can be sure that the coerced voter did not revote after casting her vote in the adversary's presence. Consequently, a tally which supports deniable revoting must even hide the number of revotes *any* voter has cast. We achieve this by anonymously checking ballots two-by-two for matching voter credentials, to avoid grouping ballots with the same credential.

As we detailed above, the ballots on the bulletin board must be shuffled before discarding superseded ballots. Because the chronological order of ballots is not retained in the shuffling process, we "memorize" the information whether a ballot was superseded with an encrypted label: Before shuffling, for each ballot we calculate an encrypted value o_i which is computed in a verifiable way by comparing voter credentials $\mathrm{pk}_i/\mathrm{pk}_j$ between the current ballot b_i and each more recent ballot b_j. The ballots themselves are not shuffled until o_i is computed for all ballots. Only then we shuffle the encrypted choice together with the o_i tag.

$$\begin{array}{rccccccccc} 12 = & 81 & \cdot & 53 & \cdot & 87 & \cdot & 48 & \bmod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 1 = & 1 & \cdot & 1 & \cdot & 1 & \cdot & 1 & \bmod N \end{array}$$

(a) Ballot is not superseded. No factor equals 1.

$$\begin{array}{rccccccccc} 25 = & 81 & \cdot & 53 & \cdot & 1 & \cdot & 48 & \bmod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 139 = & 1 & \cdot & 1 & \cdot & 139 & \cdot & 1 & \bmod N \end{array}$$

(b) Ballot is superseded once. One factor equals 1.

Fig. 1: One cannot tell from a product if one of the factors was $= 1$. To exploit the homomorphic property of the encryption, we swap the encryption of an arbitrary number with an encryption of a 1 and vice versa. Now one can tell from the product whether one of the factors was $\neq 1$.

We briefly sketch our construction. To make the duplicate test private, we mark each ballot with the encrypted information o_i whether there is another ballot that supersedes it. We compute o_i by comparing each ballot with each subsequent ballot. To do so in a publicly verifiable and private manner, we run an *encrypted plaintext equivalency test* (EPET) on the credentials. An EPET receives two ciphertexts as input, and outputs an encryption of a 1 if the contained plaintexts are equal, and the encryption of an arbitrary number otherwise. We compare ballot i with each subsequent ballot

[1] As Smith described, this attack is fatal for fake credentials, too: if the adversary can infer that a heap with 1009 ballots is discarded because of an invalid credential, the adversary knows the credential presented to him was a fake one.

by comparing its voter credential $Enc(pk_i)$ to the credentials $Enc(pk_{i+1})$, $Enc(pk_{i+2})$, etc. using an EPET.

A ballot is superseded if and only if its voter credential is equal to that of *at least one* newer ballot. We seek to exploit the homomorphic property of the encryption to aggregate the comparison results without revealing the result of any single comparison. A single EPET result is the encryption of a 1 iff the voter credentials of the ballots match. Thus, our goal is to determine for a number of EPET results whether at least one of them encrypts a 1. We cannot exploit the homomorphy directly, since a factor of 1 does not change the product (see Figure 1). As a remedy, we "convert" the values: We swap an encryption of a 1 with an encryption of an arbitrary number and vice versa. Now the product o_i tells us whether at least one factor is not equal to 1—if so, $o_i \neq \mathsf{Enc}(1)$ and the ballot is superseded, otherwise $o_i = \mathsf{Enc}(1)$ and the ballot is counted. We stress that the content of o_i is hidden by the encryption. Before checking o_i to potentially remove a ballot, the ballot's connection to its appearance on the bulletin board on the time of casting is broken by shuffling and re-encryption.

Our protocol results in a list of ballots containing the newest, valid, *encrypted* choice of each voter, and security is proven up to this point. From there, ballots can be tallied in an arbitrary way, using standard techniques like decryption mixes or homomorphic tallying, depending on the form of the choice. Security in combination with our protocol is then given by the security of the tallying procedure. As a consequence, our protocol supports arbitrary election formats, including *single transferable vote* (STV) or write-in candidates. However, our security definitions do not take into account information which is leaked by the tally result itself, which is possibly published in the form of decrypted votes. Information leaks like names of write-in candidates or a preference order can impede a voter's privacy. This problem is shared by any voting scheme which publishes the tally result or decrypted ballots, and needs to be accounted for separately by the tallying procedure.

1.3. Authentication for Coercion-Resistant Voting Schemes

All voting schemes use some method to authenticate voters. Even in elections where everybody is eligible to vote, authentication is necessary to make sure that of each voter only one vote is counted. Voter authentication can—in principle—follow any of three paradigms: authentication using something you know, something you have, or something you are. There are two properties of the authentication mechanism which are of grave importance for any voting scheme, regardless of the paradigm used: First, the adversary must not be able to impersonate the voter (e.g., by learning her secret key). Second, the adversary must not be able to prevent the voter from authenticating (e.g., by stealing her authentication token). We call an authentication *inalienable* when both properties hold. They are necessary conditions for the incoercibility of the voter. After a ballot is cast however, authentication and thus its inalienability lose their relevance. In an election which offers revoting, ballots are only ultimately cast when the polls close. Hence, any voting scheme that offers revoting requires the inalienability of the authentication until the polls have closed.

Revoting Versus Fake Credentials. Fake credentials and revoting are complementing strategies. While handing out fake credentials protects the confidentiality of the voter's true credentials, re-casting ballots "undoes" an adversary's coercion attempts. If the adversary cannot learn whether the voter has re-cast her ballot, revoting is a valid strategy for evading coercion (see Section 1.2).

A voting scheme can only support fake credentials if it does not provide any feedback on whether the voter authenticated successfully. Consequently, the voter does not learn whether her ballot was cast correctly. As Juels et al.'s construction uses fake credentials as a countermeasure against coercion, it has this property. Since our construction is an extension of theirs, their strategy for producing fake keys also works for our scheme. On the other hand, to give feedback about the success of authentication, one can publish the list of all registered voters and their public keys before the voting period—eliminating fake credentials as a strategy for evading coercion.

The fake credential strategy is an effective means to make the authentication inalienable, thus helping to achieve coercion-resistance. For the strategy to work however, the adversary must not take total control over the voter's actions, even if only temporarily. On the other hand, the revoting

strategy requires an inalienable authentication, but achieves a form of coercion-resistance that is robust even against temporary-but-perfect corruption.

1.4. Organization of this Paper

After discussing related work in Section 1.5, we introduce preliminaries and our assumptions in Section 2. In Section 3, we adapt the security notions of Juels et al. [Juels et al. 2010] to model deniable revoting. We present our voting scheme and prove its security in Section 4.

1.5. Related Work

The importance, advantages and challenges of revoting were discussed by Volkamer and Grimm [Volkamer and Grimm 2006], as well as by Post [Post 2010], though no solution for deniable revoting was given. Several internet voting schemes allow for revoting, not necessarily only as a defense against coercion. Neither of the voting schemes known to us uses a revoting strategy which is both deniable and publicly verifiable, while not requiring the voter to save state between votes. For example, in Kutyłowski and Zagórski's scheme [Kutyłowski and Zagórski 2007], in order to revoke her vote, the voter must reproduce the random challenge from the casting protocol. They further do not provide a formal proof of coercion-resistance for their scheme. In a similar vein, Spycher et al. [Spycher et al. 2010] propose a hybrid voting system where the voter can overrule her online vote in person. To this end, she must produce a re-encryption of the ballot formerly posted.

The election of the university president of the Université Catholique de Louvain in 2008 was conducted using an adapted version of Helios [Adida et al. 2009]. Because of the election's low risk of coercion attempts, revoting was supported mostly for convenience. Newer ballots of a voter substituted her older ballots on the bulletin board, so revoting was not deniable. The voting scheme designed for the 2011 local government elections of Norway [Gjøsteen 2010] supports non-deniable revoting. Verification is done by auditors, who can see the number of votes of the same voter. The Riigikogu Election Act demands the possibility of revoting for the Estonian election [Riigikogu 2002], but to the best of our knowledge, no deniable solution is offered [Maaten 2004].

As described in more detail above, the work of Juels et al. [Juels et al. 2005] supports revoting in principle, but since their work concentrates on fake voting credentials as a strategy to evade coercion, no deniable revoting strategy is proposed. Fake voting credentials are an important alternative approach for achieving coercion-resistance. After the work of Juels et al. [Juels et al. 2005] and its implementation Civitas [Myers et al. 2008], two password-based voting schemes, Selections [Clark and Hengartner 2011a; 2011b] and Cobra [Essex et al. 2012], were published. They use panic passwords as fake credentials, which can easily be created by a human. Selections is proven secure in an adapted version of the model introduced by Juels et al. [Juels et al. 2010]. Verifiable revoting is possible in both Selections and Cobra, but the number of votes cast with the same credential, or the fact that a certain ballot has been overwritten, is not hidden. Efficiency improvements of Juels et al.'s voting scheme [Juels et al. 2010] were proposed by Arajo et al. [Araújo et al. 2010], using another form of fake credential, and by Smith [Smith 2005], who introduced a faster removal procedure of duplicate ballots. Revoting is supported, but not deniable. The fake credential approach overcomes the problem of being coerced shortly before the end of the voting phase. However, the voter gets no sound confirmation upon vote casting. Especially human memory based systems like panic passwords pose the problem of accidentally using a fake credential without noticing.

By introducing the caveat coercitor [Grewal et al. 2013], Grewal et al. relaxed the requirement of coercion-resistance to coercion-evidence: a voter can be coerced, but the coercion is detectable. Their work addresses the problem of silent coercion, where the voter loses her credential to the adversary without noticing. With their approach, changing one's mind and overwriting a legit ballot is not possible (it would be recognized as a coercion attempt). Our work, in contrast, does not investigate the problem of silent coercion.

Various definitions of coercion-resistance and verifiability of e-voting schemes exist in the literature, see for example [Moran and Naor 2006; Delaune et al. 2009; Küsters et al. 2012; Unruh and Müller-Quade 2010; Canetti and Gennaro 1996] for an incomplete list. Several of them are

also applicable to our voting scheme. However, because our work aims to extend the work of Juels et al. [Juels et al. 2010], we stay close to their model for better comparability.

2. MODEL AND ASSUMPTIONS

2.1. Preliminaries and Notation

We give an overview of the notation we use in the definitions and in our protocol.

Our protocol is divided into five phases: A setup phase, a registration phase, a publication phase, the voting phase, and finally a tallying phase. We describe the phases in detail in Section 4.3. The tallying authority gets a key pair $(PK_{\mathcal{T}}, SK_{\mathcal{T}})$ during the setup phase. The secret key $SK_{\mathcal{T}}$ is shared among the servers which make up the tallying authority. In the registration phase a registrar $\mathcal{R}$ creates a public key secret key pair (pk_i, sk_i) for each voter.

Let b_i denote a ballot which is published on a bulletin board $\mathcal{BB}$, where $i \in \{1,\ldots,n\}$, let $L[i]$ denote the ith entry of a list L, and let ts_i denote the point in time when b_i has been published. The ballot represents the voter's choice $\beta_i \in \mathcal{C} = \{c_1,\ldots,c_{n_C}\}$. We describe the ballot in detail in Section 4.3. Overall n_V voters participate in the election. Let n_A denote the number of voters which are completely controlled by the adversary $\mathcal{A}$. Considering the adversary tries to coerce exactly one voter, there are $n_U = n_V - n_A - 1$ voters which add noise to the tally $\mathbf{X}$. The noise (choices of these voters) are defined by the distribution D_{n_U,n_C} which we describe in Section 3.

We use different functions of building blocks like encryption $\mathsf{Enc}(x)$, decryption $\mathsf{Dec}(c)$, verifiable shuffles $\mathsf{Shuffle}(\mathsf{L})$, (encrypted) plaintext equivalence tests (E)PET, and more. We describe them in Section 4.1 and Section 4.2.

In the experiments $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$, $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$, $\mathbf{Exp}^{revoting\text{-}c\text{-}resist}_{ES,\mathcal{A},H}$, $\mathbf{Exp}^{revoting\text{-}c\text{-}resist\text{-}ideal}_{ES,\mathcal{A},H}$ we use security parameters k_1 for the registration, k_2 for the voting function, and k_3 for the tallying process.

2.2. Attack Model

The adversary may corrupt a minority of the servers forming the tallying authority. He may also corrupt a number of voters and act in their place. Then, the adversary selects one (uncorrupted) voter he tries to coerce. In contrast to the model of Juels et al., we allow the coercion to be perfect—the coerced voter does exactly as told by the adversary. The adversary does not learn the voter's secret key, however. (See Section 2.3 for details on this assumption.)

Further, we require all adversarial control of the coerced voter, including direct observation, to end before the polls close. More concretely, we enable the coerced voter to re-cast her vote in secret. Intuitively, if the adversary cannot infer whether the coerced voter has re-cast her vote after his coercion attempt, we say the scheme offers *deniable revoting*. Clearly, revoting offers no protection against adversaries who observe or control the voter until after polls have closed.

2.3. Assumptions

We rely on various assumptions for our construction.

— Voter List. We assume an accepted voter list, i.e., there is a general agreement on who is allowed to vote.
— Bulletin Board. As is common for electronic voting schemes, we assume a publicly accessible append-only bulletin board. Anyone can read and append to the bulletin board, but no party can remove data from it.
— Authentic Timestamps. To ensure the correct order of votes, we rely on correct timestamps. Without loss of generality, we further assume that timestamps are unique.
— Anonymous Channel for Voters. To cast a vote, voters post their ballot to the bulletin board during the casting phase. For the votes to remain anonymous, we assume an anonymous channel the voters have access to. As stated by Juels et al. [Juels et al. 2010], anonymous channels can be realized with mix-nets (see Section 4.1.1).
— Inalienable Secret Credentials. In our construction, with each voter we associate a secret that she uses to authenticate her vote. The secret is used as a signature key for ensuring the integrity of

the ballot. Also, the corresponding public key is contained on the ballot in encrypted form. In a practical realization of our scheme, one would have to take special precautions to guarantee that the adversary can neither learn the secret credential nor deny the voter access to it. Realizing this assumption is outside the scope of this work. However, we point out that the assumption can be argued for: The election secret could be stored on a tamper-proof device that also serves a different function (e.g. a federal identity card), such that the voter cannot reasonably be without it. Voters would have reservations against handing out such a device (see also Section 1.3).

3. SECURITY NOTIONS

To model voter behavior when one is allowed to re-cast one's vote, we define D_{n_U,n_C} to be a distribution over all vectors over all (bounded) series of all candidates, including abstentions and invalid votes. Let ϕ denote the null ballot (abstention) and λ an invalid ballot. Then D_{n_U,n_C} is a distribution over vectors $((\beta_1)_j,(\beta_2)_j,\ldots,(\beta_{n_U})_j),\beta_i \in (C \cup \{\phi,\lambda\})$. For technical reasons, the length of the vote series is bound by a constant, lest the length of the voter's choices exceeds the runtime of all machines involved. In practice, one can imagine this bound to be the number of nanoseconds between the start of the voting period and its end, for example.

Further, we define the number of uncertain votes, i.e. votes cast by voters not under adversarial control or coercion, as $n_U := n_V - n_A - 1$. Let $\leftarrow$ denote the assignment operation, and $\Leftarrow$ the append operation. For any experiment $\mathbf{Exp}^x_{\mathcal{A}}$, where $x \in \{corr, ver, revoting\text{-}c\text{-}resist, revoting\text{-}c\text{-}resist\text{-}ideal\}$, we define the adversary's success probability as $\mathbf{Succ}^x_{\mathcal{A}}(k_1,k_2,k_3,n_V,n_C) := Pr[\mathbf{Exp}^x_{\mathcal{A}}(k_1,k_2,k_3,n_V,n_C) = 1]$. All algorithms are implicitly assumed to be PPT, i.e., run in probabilistic polynomial time for some fixed polynomial. A function $f(k)$ is *negligible in parameter k* if for all positive integers c there is an l_c such that $f(k) < k^{-c}$ for all $k > l_c$.

3.1. Correctness

Our notion of correctness follows that of Juels et al. We model voters not as posting one ballot, but a series of ballots. The adversary may corrupt a number of voters and vote in their place. We call a tally correct when, regardless of the behavior of corrupted parties, the last ballot, and only the last ballot, of each voter is counted.

See Figure 2 for Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$. A voting protocol is *correct* if $\mathbf{Succ}^{corr}_{ES,\mathcal{A}}(k_1,k_2,k_3,n_V,n_C)$ is negligible in k_1,k_2,k_3 for any adversary $\mathcal{A}$.

Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$

1 $\{(sk_i,pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
2 $V \leftarrow \mathcal{A}(\{pk_i\}_{i_1}^{n_V}$, "choose controlled voter set");
3 $\{\beta_{i,j}\}_{i\notin V} \leftarrow \mathcal{A}$("choose votes from uncontrolled voters");
4 $\mathcal{BB} \Leftarrow \{\texttt{vote}(sk_i,PK_{\mathcal{T}},\mathcal{BB},n_C,\beta_{i,0},k_2)\}_{i\notin V}$;
5 $(\mathbf{X},P) \leftarrow \texttt{tally}(SK_{\mathcal{T}},\mathcal{BB},n_C,\{pk_i\}_{i_1}^{n_V},k_3)$;
6 $\mathcal{BB} \Leftarrow \mathcal{A}$("cast ballots", $\mathcal{BB}$)
7 $(\mathbf{X}',P') \leftarrow \texttt{tally}(SK_{\mathcal{T}},\mathcal{BB},n_C,\{pk_i\}_{i_1}^{n_V},k_3)$;
8 **if** $\texttt{verify}(PK_{\mathcal{T}},\mathcal{BB},n_C,\mathbf{X}',P') = 1$ **and** $(\{\beta_{i,0} \not\subset \langle\mathbf{X}'\rangle$ **or** $|\langle\mathbf{X}'\rangle| - |\langle\mathbf{X}\rangle| > |V|)$ **then**
output 1;
else
output 0;

Fig. 2: Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$. A tallying scheme is correct if the last, and only the last, vote of legitimate voters is counted.

3.2. Verifiability

We adopt Juels et al.'s notion of verifiability. See Figure 3 for Experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$. A voting protocol is *verifiable* if $\mathbf{Succ}^{ver}_{ES,\mathcal{A}}(k_1,k_2,k_3,n_V,n_C)$ is negligible in k_1,k_2,k_3 for any adversary $\mathcal{A}$.

Experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$

$\{(sk_i, pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
$(\mathcal{BB}, \mathbf{X}, P) \leftarrow \mathcal{A}(SK_{\mathcal{T}}, \{(sk_i, pk_i)\}_{i=1}^{n_V}$, "forge election");
$(\mathbf{X}', P') \leftarrow \texttt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$;
if $\mathbf{X} \neq \mathbf{X}'$ **and** $\texttt{verify}(PK_{\mathcal{T}}, \mathcal{BB}, n_C, \mathbf{X}, P) = 1$ **then**
  output 1;
else
  output 0;

Fig. 3: Experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$. A tallying scheme is verifiable if $\mathcal{A}$ is not able to state a wrong tally with a correct proof.

3.3. Deniability of Revoting

Similarly to how Juels, Catalano, and Jakobsson model coercion-resistance, we model the deniability of revoting as two experiments following the "real-world-ideal-world" paradigm: Even in a "flawless" voting scheme some information leaks—the total number of cast ballots, the tally, and possibly even who participated in the vote. When measuring the advantage of the adversary, we seek to factor out "unavoidable" leaks of information. To this end we define an *ideal* experiment ($\mathbf{Exp}^{revoting\text{-}c\text{-}resist\text{-}ideal}_{ES,\mathcal{A},H}$) that captures all such leaks. The *real* experiment ($\mathbf{Exp}^{revoting\text{-}c\text{-}resist}_{ES,\mathcal{A},H}$) captures the concrete voting protocol. We then examine the difference of the probability of success of an adversary in the ideal world versus an adversary in the real protocol.

In the "real world" experiment, an election is carried out as specified by the protocol. The adversary can corrupt and thus completely control a set of voters. We model this by handing the secret keys of corrupted voters to the adversary. Uncorrupted voters cast their ballots according to the distribution D_{n_U,n_C}. Further, the adversary may select one (uncorrupted) voter as his coercive target. As in the original definition by Juels et al., an extension to a simultaneous coercion of more than one voter is straightforward. The voter does not carry out an evasion strategy in our experiment, i.e. the adversary receives the coerced voter's secret key and may cast votes in the voter's stead. (Note that we actually assume the adversary *cannot* learn the coerced voter's secret key in reality (see Section 2.3). We model time-limited, perfect coercion by giving the adversary access to the secret key, and not accepting further output later.) Then, a bit b is flipped. If $b = 0$, the coerced voter evades coercion by revoting. If $b = 1$, the voter submits to the coercion, i.e., does nothing. After all votes are cast, a tally and corresponding proofs are computed and handed to the adversary. The adversary then guesses b. See Figure 4 for Experiment $\mathbf{Exp}^{revoting\text{-}c\text{-}resist}_{ES,\mathcal{A},H}$.

In the ideal experiment, we make use of a function `ideal-tally`, which represents an ideal tallying process. Its working depends on the challenge bit b. If $b = 0$ (voter evades coercion) it counts the coercive target voter's vote. Otherwise (voter submits to coercion) it counts the adversary's choice for the coercive target voter's vote. Further, the ideal adversary is not supplied with the contents of the bulletin board, but only its length l, as well as the election result $\mathbf{X}$. If the coercive target voter evades coercion, the reported length of the bulletin board is increased by the length of one ballot. (Note that the overall number of revotes can always be inferred if the number of cast ballots and the number of counted ballots is visible.) See Figure 5 for Experiment $\mathbf{Exp}^{revoting\text{-}c\text{-}resist\text{-}ideal}_{ES,\mathcal{A},H}$.

Experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1, k_2, k_3, n_V, n_A, n_C)$

$V \leftarrow \mathcal{A}$(voter names, "control voters");
$\{(sk_i, pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
$(j, \beta) \leftarrow \mathcal{A}(\{sk_i\}_{i\in V}$, "set target voter and vote");
if $|V| \neq n_A$ **or** $j \notin \{1,2,\ldots,n_v\} - V$ **or** $\beta \notin \{1,2,\ldots,n_C\} \cup \emptyset$ **then**
  output 0;
$\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_i\}_{i\notin V}, PK_{\mathcal{T}}, n_C, D_{n_U,n_C}, k_2)$;
$\mathcal{BB} \Leftarrow \mathcal{A}(sk_j, \mathcal{BB}$, "cast ballots");
$b \in_U \{0,1\}$;
if $b = 0$ **then**
  $\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_j\}, PK_{\mathcal{T}}, n_C, D_{n_U,n_C}, k_2)$;
$(\mathbf{X}, P) \leftarrow \texttt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$;
$b' \leftarrow \mathcal{A}(\mathbf{X}, P, \mathcal{BB}$, "guess b");
if $b' = b$ **then**
  output 1;
else
  output 0;

Fig. 4: Experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$. This experiment describes the possibilities of an adversary $\mathcal{A}$ in the "real world", including the tally result.

Experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}(k_1, k_2, k_3, n_V, n_A, n_C)$

$V \leftarrow \mathcal{A}'$(voter names, "control voters");
$\{(sk_i, pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
$(j, \beta) \leftarrow \mathcal{A}'$("set target voter and vote");
if $|V| \neq n_A$ **or** $j \notin \{1,2,\ldots,n_v\} - V$ **or** $\beta \notin \{1,2,\ldots,n_C\} \cup \emptyset$ **then**
  output 0;
$b \in_U \{0,1\}$;
$\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_i\}_{i\notin V}, PK_{\mathcal{T}}, n_C, D_{n_U,n_C}, k_2)$;
$\mathcal{BB} \Leftarrow \mathcal{A}'(sk_j, |\mathcal{BB}|$, "cast ballots");
$l \Leftarrow |\mathcal{BB}|$;
if $b = 0$ **then**
  $l \Leftarrow |l| + 1$;
$(\mathbf{X}, P) \leftarrow \texttt{ideal-tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3, b)$;
$b' \leftarrow \mathcal{A}'(\mathbf{X}, l$, "guess b");
if $b' = b$ **then**
  output 1;
else
  output 0;

Fig. 5: Experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}$. This experiments describes the possibilities of an adversary $\mathcal{A}'$, based on the tally result. The success of $\mathcal{A}'$ normalizes the success of $\mathcal{A}$.

We define the *advantage* of the adversary $\mathcal{A}$ as $\mathbf{Adv}^{\textit{revoting}}_{\mathcal{A}}(k_1,k_2,k_3,n_V,n_A,n_C) :=$

$$\mathbf{Succ}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1,k_2,k_3,n_V,n_A,n_C) - \mathbf{Succ}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(k_1,k_2,k_3,n_V,n_A,n_C).$$

A voting scheme features *deniable revoting* if $\mathbf{Adv}^{\text{revoting}}_{\mathcal{A}}$ is negligible in k_1,k_2,k_3 for any adversary.

4. A COERCION-RESISTANT VOTING PROTOCOL WITH DENIABLE REVOTING

In this section, we first introduce several building blocks used in our protocol. We then describe our construction and prove its security according to the definitions introduced in the previous section.

4.1. Black-box Ideal Functionalities

We use several primitives as blackboxes and assume the existence of an efficient realization. Though we suggest efficient instantiations where possible, we focus on our general approach for achieving deniable revoting. We see our work as a proof of concept.

4.1.1. Verifiable Secret Shuffle. A verifiable shuffle computes a function $\mathsf{Shuffle}(\mathsf{C}) \mapsto (\mathsf{C}', \mathsf{P})$, which gets as input a list $C := [c_1, \ldots, c_n]$ of randomizable ciphertexts. Its output consists of a list $C' := [c'_{\pi(1)}, \ldots, c'_{\pi(n)}]$, where π is a random permutation and $c'_{\pi(i)}$ is a re-encryption of c_i for $i = 1, \ldots, n$, and a proof P of correctness of the shuffle.

We assume our shuffles are *secret* and *verifiable*: secrecy implies it is infeasible to link an input ciphertext c_i to its corresponding output ciphertext $c'_{\pi(i)}$ better than guessing, and verifiability requires a proof P that C is indeed a permutation (with re-encryption) of C'.

We further assume our shuffles are *distributable* among several servers, and speak of a *mix-net* if a shuffle is distributed. Examples for verifiable secret shuffles and mix-nets are [Neff 2001; Sako and Kilian 1995; Groth 2002; Khazaei et al. 2012; Abe and Hoshino 2001; Golle et al. 2004].

4.1.2. EUF-CMA Secure Digital Signatures. In our voting scheme, voters use unforgeable signatures for proving their eligibility. A signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is called *existentially unforgeable against adaptive chosen message attacks (EUF-CMA)* [Goldwasser et al. 1988], if $Pr[(vk, sk) \leftarrow \mathsf{KeyGen}(1_k), (\sigma, m) \leftarrow \mathcal{A}^{O_{Sign(.)}}(vk) : \mathsf{Verify}(vk, \sigma, m) = 1]$ is negligible in security parameter k, where $O_{Sign(.)}$ is a signature oracle which on input of a message m_i ouputs a valid signature $\sigma_i = \mathsf{Sign}(sk, m_i)$ with $\mathsf{Verify}(vk, \sigma_i, m_i) = 1$, and m has never been queried to $O_{Sign(.)}$. We require our signatures to allow for an efficient zero-knowledge proof of signature knowledge.

4.1.3. Non-Interactive Zero-Knowledge Proofs (of Knowledge). We make use of Non-Interactive Zero-Knowledge Proofs (NIZK) and non-Interactive Zero-Knowledge Proofs of Knowledge (NIZKPoK) in our construction. NIZK and NIZKPoK exist for arbitrary NP statements. Correct decryption can be proven by using the Chaum-Pedersen-Protocol [Chaum and Pedersen 1993] as a proof of logarithm equivalence. Logarithm knowledge can be proven with the Schorr-protocol [Schnorr 1991]. A proof of signature knowledge was introduced by Camenisch et al. [Camenisch and Stadler 1997]. Structure-preserving signatures [Abe et al. 2010] with Groth-Sahai proofs [Groth and Sahai 2008] are an alternative that does not rely on a random oracle.

4.2. Building Blocks

4.2.1. Modified Elgamal Encryption (m-Elgamal). M-Elgamal is a modification of the Elgamal encryption scheme [Elgamal 1985], used by Juels et al. [Juels et al. 2010]. Given a multiplicative cyclic group G of prime order p in which the decisional Diffie-Hellman problem is hard, and generators $g_1, g_2 \in G$, for key generation choose random $x_1, x_2 \in \mathbb{Z}_p$, and output secret key (x_1, x_2) and public key $y := g_1^{x_1} g_2^{x_2}$. To encrypt a message m with public key y, choose $r \in \mathbb{Z}_p$ at random, and output the ciphertext $c = \mathsf{Enc}(m, y) := (g_1^r, g_2^r, y^r m)$. To decrypt a ciphertext $c = (A, B, C)$ with secret key (x_1, x_2), compute $m = \mathsf{Dec}(c, (x_1, x_2)) := A^{-x_1} B^{-x_2} C$. To simplify notation, we write $\mathsf{Enc}(m)$ or $\mathsf{Dec}(c)$ if the keys are clear from context. Decryption can be distributed using the construction of Cramer et al. [Cramer et al. 1997]. Ciphertexts created with m-Elgamal are multiplicatively homomorphic and randomizable by multiplying with an encryption of 1. A proof of encryption of a certain plaintext can be achieved by publication of the encryption randomness followed by randomization. Correct decryption can be proven with the Chaum-Pedersen-protocol [Chaum and Pedersen 1993].

4.2.2. Plaintext Equality Test. Plaintext equality tests (PET) were introduced by Juels et al. [Jakobsson and Juels 2000; Juels et al. 2010]. They decide whether two ciphertexts contain the same plaintext. We denote this by a function $\mathsf{Pet}(c_1, c_2, t) \mapsto (b, \Pi)$ which gets as input two ciphertexts c_1 and c_2 and a decryption trapdoor t. Its output is a bit b with $b = 1$ if c_1 and c_2 contain the same plaintext, and $b = 0$ otherwise, as well as a proof Π of correctness. In our protocol, we perform computations on encrypted PET results. We define *encrypted plaintext equality tests (EPETs)* denoted by a function $\mathsf{Epet}(c_1, c_2) \mapsto (c, \Pi_c)$ which outputs an encryption of 1 if c_1 and c_2 contain the same plaintext, and an encryption of a random number, if the plaintexts are different, as well as a proof Π_c of correctness. EPETs can be computed without a trapdoor. We require PETs to be distributable.

(Encrypted) Plaintext Equality Test for Homomorphic Encryption. Juels et al. [Jakobsson and Juels 2000] introduced a plaintext equality test for Elgamal encryption. We generalize their result to a PET for multiplicatively homomorphic encryption which can also be used as an EPET. Instantiating it with Elgamal or m-Elgamal allows a distributed execution of the PET.

Let *Enc* denote a multiplicatively homomorphic encryption function, i.e., $Enc(m_1) \cdot Enc(m_2) = Enc(m_1 \cdot m_2)$, and *Dec* its decryption function. We can perform a PET on two ciphertexts $c_1 = Enc(m_1)$ and $c_2 = Enc(m_2)$ for plaintexts m_1 and m_2 as follows:

Algorithm $\mathsf{Epet}(\mathsf{c_1}, \mathsf{c_2})$: On input of two ciphertexts c_1 and c_2, choose r at random and compute $c_{diff} := (c_1/c_2)^r$ with a NIZKPoK Π_c of an r such that $c_{diff} = (c_1/c_2)^r$, for example by using the Chaum-Pedersen-protocol [Chaum and Pedersen 1993]. Exploiting the homomorphic property of the encryption, we have $c_{diff} = (c_1/c_2)^r = Enc((m_1/m_2)^r) = 1^r = 1$ if $m_1 = m_2$ (or $r \equiv 0 \mod Ord(G)$, see below). By outputting c_{diff} and the proof Π_c, this scheme can be used as an EPET.

Algorithm $\mathsf{Pet}(\mathsf{c_1}, \mathsf{c_2}, \mathsf{t})$: To make a PET out of the EPET, first compute $(c_{diff}, \Pi_c) := \mathsf{Epet}(c_1, c_2)$, then decrypt c_{diff} using decryption key t, with a proof Π_d of correct decryption. Output 1 if $Dec(c_{diff}) = 1$, and 0 otherwise, as well as the proof $\Pi := (\Pi_c, \Pi_d)$. If $m_1 \neq m_2$, c_{diff} is a random number because of the mask r, and can be revealed in order to prove correctness of the result. Also, if it is not clear from the form of c_{diff} that $r \neq 0 \mod \mathrm{Ord}(G)$, r can be opened if $Dec(c_{diff}) = 1$.

4.3. Our Construction

We divide elections into five phases: setup, registration, setup publication, voting, and tallying.

(1) Setup. The tellers create a joint public encryption key $PK_{\mathcal{T}}$ and a shared secret decryption key sk_T for threshold decryption for the homomorphic encryption scheme m-Elgamal introduced in Section 4.2.1. Let $\mathsf{Enc}(.)$ denote its encryption function.

(2) Registration. Each voter V_j obtains their credential for voting, i.e., a key pair (pk_j, sk_j) for an EUF-CMA secure signature scheme (see Section 4.1.2). We assume that an efficient non-interactive zero-knowledge proof of knowledge of a signature exists for this signature scheme. The public verification key obtained here also acts as identification of the voter. For each voter, an entry $(ID, \mathsf{Enc}(pk_j))$ is stored in a List L_0, where ID encodes the voter's name. The encryption $\mathsf{Enc}(pk_j)$ is computed using the teller's public key $PK_{\mathcal{T}}$.

(3) Setup publication. A candidate-slate $\mathcal{C} = (c_1, \ldots, c_l)$ is published on the public bulletin board. It contains the electable candidates $c_1, \ldots, c_l$. The list L_0 is published and serves as a list of eligible voters and their encrypted public verification keys. The public key $PK_{\mathcal{T}}$ of the tellers is published as well.

(4) Voting. Voters create ballots and submit them to the bulletin board. A ballot b_i is a three-tuple as explained below. Encrypted parts are encryptions under the public key of the tellers:

$$b_i = (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i),\ NIZKPoK$$

Here, $\mathsf{Enc}(\beta_i)$ is an encryption of the choice $\beta_i \in \mathcal{C}$. We assume this for the simplicity of our description. To support arbitrary choices like single transferable vote (STV), we can alternatively let the choice be a function of the candidate slate and the voter's intention. Furthermore, $\mathsf{Enc}(pk_i)$ is an encryption of the voter's public key pk_i, and ts_i is a timestamp (see Section 2.3) created right before the ballot is cast. The timestamp is not encrypted. In addition to the ballot itself,

the voter submits a non-interactive zero-knowledge proof of knowledge *NIZKPoK* to prove that the ballot has been cast with her consent. The NIZKPoK proves knowledge of a signature of $(\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$ w.r.t. the public key encrypted in the ballot. (Directly attaching the signature itself would reveal how many votes a single voter has cast.) Stated more formally, the voter proves knowledge of a signature σ_i with

$$\mathtt{verify}(\sigma_i, (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i), pk_i') = 1 \ \wedge \ pk_i' = pk_i.$$

(5) Tallying. Before the ballots can be tallied, all superseded ballots have to be sorted out. This procedure is described in detail in the next section. After all ballots with invalid proofs of signature knowledge, all superseded ballots, and all ballots with invalid credentials have been omitted, the remaining ballots can be tallied with standard techniques, for example by a decryption mix.

4.4. Sorting Out Superseded Ballots

In this section, we describe how to mark all superseded ballots. A ballot is superseded when a more recent ballot of the same voter—that is, a ballot with a newer timestamp and matching voter credential—is posted to the bulletin board. Our method protects the privacy of the voter and is also publicly verifiable.

In four steps, the tallying authority computes a shuffled list of all valid ballots without their timestamps. They will be re-encrypted and in a random order. The resulting list will only contain the last votes cast by eligible voters. W.l.o.g. we assume all submitted choices are valid.

(1) **Encrypted Tests Whether Two Ballots Are From The Same Voter** The procedure starts off with a list of all "well-formed" ballots on the bulletin board, i.e. all ballots with a valid NIZKPoK as described above. After it has been checked, the NIZKPoK is omitted in further steps. The list is sorted according to the timestamp in ascending order. For each ballot b_i, the tallying authority tests if the encrypted public key pk_i of b_i matches the public key pk_j of any newer ballot b_j (see Section 4.4.1): for each ballot b_j ($i < j < n$), the tallying authority performs distributed EPETs to obtain $(d_{i,j}, \Pi_{i,j}) := \mathsf{Epet}(Enc(pk_i), Enc(pk_j))$. The encrypted differences $d_{i,j}$ and the proofs $\Pi_{i,j}$ of correctness of the EPET-result are published. ($d_{i,j} = \mathsf{Enc}(1)$ iff $pk_i = pk_j$.)

(2) **Marking Ballots As Superseded** The tallying authority performs a verifiable conversion on all computed differences. If $d_{i,j}$ is an encryption of 1, replace it by an encryption of a random number. Else, replace it with an encryption of 1. Differences are detached from their corresponding ballots before conversion, and later sorted back. The details of this step are described in Section 4.4.2. For each ballot, the tallying authority aggregates all converted $d'_{i,j}$ by multiplying them, exploiting the homomorphic property of the encryption: $o_i := \prod_j d'_{i,j}$

(3) **Omit Superseded Ballots** The tallying authority jointly compares o_i with $\mathsf{Enc}(1)$ for each ballot b_i. It omits all ballots b_i with $\mathsf{Epet}(\mathsf{o_i}, \mathsf{Enc}(1)) \neq 1$ from future computations. (Those are the ballots that have been superseded by a more recent one of the same voter.) The last ballot is never superseded.

(4) **Omit Ballots With Invalid Credentials** Before tallying, the tallying authority checks the voter credentials by verifying that each $\mathsf{Enc}(pk_i)$ has a corresponding entry in the published list of the encrypted public keys of eligible voters. Similarly to the technique of Juels et al. [Juels et al. 2010], the tallying authority shuffles the encrypted public keys of the list L_0, and performs a PET with the encrypted public key of each ballot.

We now describe the four steps in detail.

4.4.1. Encrypted Tests Whether Two Ballots Are From The Same Voter. In the first step of the process, for each pair of ballots (b_i, b_j) with $j > i$, the tallying authority applies EPETs (see Section 4.2.2) to the encrypted public keys of the voter. While, technically, this step can be performed during the tally phase, we propose to directly perform it during the casting phase:

When a ballot $b_i = ((\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$ is cast, the tallying authority checks its NIZKPoK and discards the ballot if the proof is invalid, i.e. the ballot is marked as invalid and not considered

in further computations, but remains on the bulletin board for public verification. Otherwise, the tallying authority jointly runs an EPET on the encrypted public key of b_i and those of all already cast ballots b_j, to obtain values $(d_{i,j}, \Pi_{i,j}) = \mathsf{Epet}(\mathsf{Enc}(pk_i), \mathsf{Enc}(pk_j))$. All encrypted differences $d_{i,j}$ are stored alongside b_j as indicated in Figure 6, defining a list L with entries $L[i] = (b_i, (d_{i,i+1}, \ldots, d_{i,n}))$.

After the casting phase, if n well-formed ballots have been cast, b_i has $n-i$ associated differences.

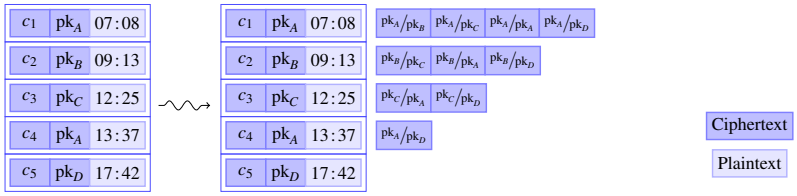


Fig. 6: Encrypted Plaintext Equality Tests (EPETs) are performed for each pair of ballots (b_i, b_j) with $j > i$ by the tallying authority: If $pk_i = pk_j$ the result is an encrypted 1, otherwise it is an encrypted random value. We denote an encrypted value by a box with a dark background and an unencrypted value by a box with a lighter background. In this example, the third fraction in the first row $(\mathrm{pk}_A/\mathrm{pk}_A)^r$ divides identical credentials, hence ballot 4 supersedes ballot 1.

4.4.2. Marking Ballots as Superseded. Before computing the supersede mark o_i for each ballot, the differences $d_{i,j}$ computed during the voting phase are converted as indicated by the mapping

$$\mathsf{Enc}(x) \mapsto \begin{cases} \mathsf{Enc}(r) & \text{if } x = 1 \\ \mathsf{Enc}(1) & \text{if } x \neq 1 \end{cases}$$

The value r can be any fixed value other than 1 (2, for example) or a number drawn at random.

To compute the mapping, the tallying authority creates a shuffled and randomized list of all encrypted differences. To this end, each entry $d_{i,j}$ is associated an encrypted tag $\mathsf{Enc}(\mathrm{ts}_i)$ (see Figure 7).

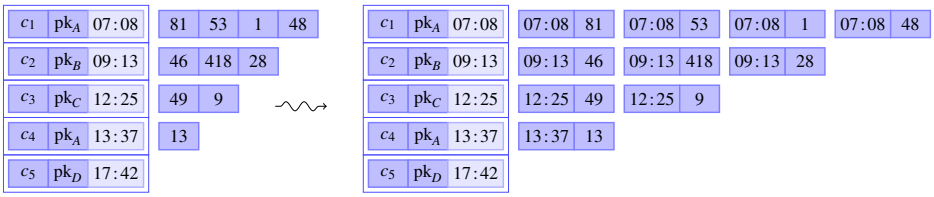


Fig. 7: The fractions from the previous step are complemented by the encrypted timestamp from the ballot, forming tuples $(\mathsf{Enc}(ts_i), d_{i,j})$. For the sake of clarity, we evaluated the fractions from Figure 6 to exemplary values. The superseded first ballot thus has a "1" associated with it.

The list of all tuples $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j})$ is then re-randomized and reordered, using a verifiable secret shuffle. After shuffling, the differences are converted (see Figure 8). To convert $d_{i,j}$ to $d'_{i,j}$ we perform a multi-party computation that involves a "coordinator" and the voting authority. In this step, the authority's task is to decrypt each $d_{i,j}$ it receives from the coordinator and return either an encryption of a 1 or of another (random or fixed) number, according to the map above. The task of the coordinator is to send fake differences or real ones to the authority. He has to make sure that the authority does not learn which conversion is real. Therefore, each element is randomized before and after each conversion by the coordinator. The randomized version of the output of each "real" conversion is fed to the authority to be converted a second time and get a fake difference. The second conversion is necessary to hide how many of the $d_{i,j}$ contain a 1. All (real and fake) differences are sent to the authority in random order. The coordinator and the authority must not collude.

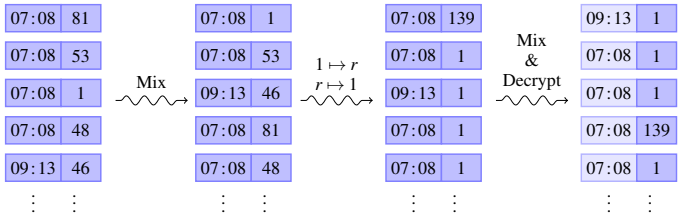


Fig. 8: Converting $d_{i,j}$ to $d'_{i,j}$: if $\mathsf{Dec}(d_{i,j}) = 1$ replace $d_{i,j}$ by $\mathsf{Enc}(r)$ ($r \neq 1$) and by $\mathsf{Enc}(1)$ else. The special "superseded" value 1 has been converted to an arbitrary number, while all other values are mapped to 1.

Irrespective of how the entries $d_{i,j}$ are converted to $d'_{i,j}$, their correctness can easily be proven: either $\mathsf{Pet}(d_{i,j} \cdot d'_{i,j},\ d_{i,j}) = 1$ or $\mathsf{Pet}(d_{i,j} \cdot d'_{i,j},\ d'_{i,j}) = 1$, never both. To prove the correctness of the inversion without revealing the values, we use a verifiable shuffle on $(d_{i,j},\ d'_{i,j})$ to obtain (a,b). We then check whether $\mathsf{Pet}(a \cdot b,\ a) = 1$ or $\mathsf{Pet}(a \cdot b,\ b) = 1$ (exclusively). See Appendix A for a compact description of the procedure.

After a second shuffling step, the tags are decrypted, and the encrypted, converted differences are sorted back to their ballot of origin (see step one in Figure 9). All steps can be verified by the public. Then the tallying authority computes the homomorphic product over all the associated marks of each ballot (see step two in Figure 9): for each i compute

$$o_i := \prod_{j=\min(i+1,n)}^{n} d'_{i,j}.$$

Observe that if $\prod d'_{i,j}$ contains only encryptions of 1, it is itself an encryption of a 1, whereas if it has a factor $\neq 1$, it is itself an encryption of a number $\neq 1$ with overwhelming probability.

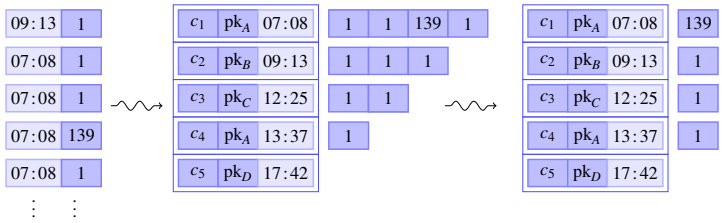


Fig. 9: The encrypted, converted differences are sorted back to their ballot of origin. Afterwards o_i is computed as the homomorphic product over all the associated marks of ballot b_i. Note that we arranged the $d'_{i,j}$ values in the same order as their preimages. In a practical realization of our scheme this would only coincidentally be the case. Ballots now have an encrypted "tag" that tells whether they are superseded by a more recent ballot.

4.4.3. Omit Superseded Ballots. Before any further processing, particularly before checking if $o_i = Enc(1)$, all ballots are weeded and shuffled. Only the encrypted choice of the voter, his encrypted public key, and the mark o_i are kept. Therefore, the tallying authority forms a list L_W with entries $L_W[i] := (b'_i, o_i)$, where $b'_i := (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i))$ if $b_i = (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$. The tallying authority computes and publishes $(L'_W, \Pi) := \mathsf{Shuffle}(\mathsf{L_W})$ and then jointly compares o_i with $\mathsf{Enc}(1)$ in L'_W using a PET, and publishes all proofs. Only ballots b_i with $o_i = \mathsf{Enc}(1)$ and the last ballot are kept, the others are marked as discarded.

4.4.4. Omit Ballots With Invalid Credentials. Finally, the validity of the public keys is checked. Note that, at this point, only one ballot per credential remains. Equally to the method of Juels et al., the tallying authority shuffles the encrypted public keys of L_0 and performs a joint PET with the

encrypted public key of each ballot. Ballots with valid public keys are retained. The encrypted choices of the retained ballots can then be mixed and opened for tallying.

4.5. Proving the Security of our Scheme

In this section, we prove deniability of revoting in our protocol. Correctness and verifiability are addressed in Appendix B and C, respectively.

To prove the deniability of revoting, we give a reduction of the privacy of our scheme to the Decisional Diffie-Hellman (DDH) problem. More concretely, given any (successful) adversary $\mathcal{A}$ against $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$, we construct a simulator $\mathcal{S}$ which is, with non-negligible probability, able to decide for a given quadruple (g, g^a, g^b, g^c) whether $c = ab$ in the cyclic group $G = \langle g \rangle$. As in the definition, we consider only one coerced voter. An extension to the case of multiple coerced voters is straightforward—we just have accordingly many parallel instances of the DDH problem. Our proof is similar to that of Juels et al. [Juels et al. 2005]. We give a reduction from any adversary that uses the published ballots to break the incoercibility of the scheme to an adversary against the DDH assumption. To this end, we simulate the execution of experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$.

The DDH experiment $\mathbf{Exp}^{\mathsf{ddh}}_{G}$ secretly draws a bit d, if $d = 1$ sets $g^c = g^{ab}$, and to a random element $g^c \in G$ otherwise. Given a DDH challenge (g, g^a, g^b, g^c), $\mathcal{S}$ deducts two public keys. In the setup publication phase, $\mathcal{S}$ sends the first one to the adversary, while it uses the second one during the rest of the protocol. $\mathcal{S}$ deducts the following two m-Elgamal public keys:

$$(g_1 := g,\ g_2 := g^a,\ y_g := g_1^{x_1} g_2^{x_2} = g^{x_1} g^{a x_2}) \text{ and}$$

$$(h_1 := g^b,\ h_2 := g^c,\ y_h := h_1^{x_1} h_2^{x_2} = g^{b x_1} g^{c x_2}).$$

Recall that, to encrypt m using the Modified Elgamal Encryption (see Section 4.2.1) using public key (g_1, g_2, y), one chooses a random r and outputs $(g_1^r, g_2^r, y^r m)$. When $c = ab$ ($d = 1$ in the surrounding experiment), for any m there are r, r' such that $(g_1^{r'}, g_2^{r'}, y_g^{r'} m) = (h_1^r, h_2^r, y_h^r m)$. ($r' = br$, concretely.) Therefore, ciphertexts created using the above public keys have the same distribution.

When, on the other hand, $c \neq ab$ ($d = 0$), m is perfectly hidden in the encryption. A message m, encrypted with the second public key and decrypted with the first (given to $\mathcal{A}$), yields:

$$(h_1^r = g^{br} = g_1^{br}, h_2^r = g^{cr} = g_2^{(c/a)r},$$

$$y_h^r m = g^{rbx_1} g^{rcx_2} m = g^{rbx_1} g^{abrx_2} g^{(c-ab)rx_2} m = y_g^{br} g^{(c-ab)rx_2} m)$$

In $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$ the choices of the voters as well as their public keys are perfectly hidden in the m-Elgamal Encryption when $d = 0$. In this case the adversary's capabilities are reduced to those in experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$. To justify this assertion, we must show how the adversary's input can be faked in the $d = 0$ case with only the information from $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$ available. It is then obvious that the adversary cannot gain any additional advantage from the simulated tallying process.

In the ideal experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$ the adversary learns the total number of cast ballots and the number of valid ballots. The individual m-Elgamal encryptions the simulator outputs to the adversary hide the plaintext perfectly when $d = 0$. In this case, they are not distinguishable from any random group element and thus can be faked by random output. We must also be able to fake the quantity and structure of the data the adversary learns during the simulation. These can be determined from the difference of cast and valid ballots, and vice versa.

We state the simulation of $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$ and argue that its perfect. The simulator $\mathcal{S}$ receives a $W \in D_{n_U, n_C}$ and a challenge (g, g^a, g^b, g^c) with $c = ab$ if the random bit $d = 1$ or $c \neq ab$ if $d = 0$.

Setup. The simulator $\mathcal{S}$ chooses the secret key $SK_{\mathcal{T}} := (x_1, x_2)$, uniformly and at random. $\mathcal{S}$ also calculates the public key $PK_{\mathcal{T}} := (g^{x_1}, g^{x_2}, h = (g_1^{x_1} g_2^{x_2}) \mod p$ and sets $C = \{c_i\}_{i=1}^{n_C}$.

Registration. $\mathcal{S}$ simulates the registrar $\mathcal{R}$: $\mathcal{S}$ generates the voter credentials: $\{(pk_i, sk_i)\}_{i=1}^{n_V}$
Setup publication. $\mathcal{S}$ publishes the public key of the tally $PK_{\mathcal{T}}$, the set of candidates C, and $L_0 = \{$"voter's name i", $\mathsf{Enc}(pk_i)\}_{i=1}^{n_V}$.
Adversarial corruption. The adversary $\mathcal{A}$ selects n_A voters to corrupt. Let V denote the group of corrupted voters. He also selects a voter j for coercion. He determines the choices β for the corrupted and the coerced voters. The simulation is terminated if the selection is invalid.
Coin flip. The bit b is chosen uniformly at random: $b \in_U \{0,\ 1\}$
Honest voter simulation. For each honest voter, the simulator creates the ballots with the proofs:

$$A_0 := \{(\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i),\ NIZKPoK\}$$

$$= \{b_i := ((h_1^{r_i}, h_2^{r_i}, h_1^{r_i x_1} h_2^{r_i x_2} c_j), (h_1^{k_i}, h_2^{k_i}, h_1^{k_i x_1} h_2^{k_i x_2} pk_i), ts), NIZKPoK\}_{i=1}^{n}$$

Thereby all r_i and k_i are chosen at random in Z_q. The choices are determined in W. For creating the proofs the simulator $\mathcal{S}$ uses the voter's secret keys.
Adversarial ballot posting. The adversary calculates the ballots for each voter $v \in V$ and the voter j in the same way. We call the set of these ballots B_0. $\mathcal{A}$ posts B_0.
Tallying simulation. $\mathcal{S}$ simulates a honest tallying authority, whereby it uses the secret key $SK_{\mathcal{T}}$ from the setup step. Since the correctness of every step from each authority can be verified, any modification from an adversary can be ignored.

Proof checking. The proof of each ballot in A_0 and B_0 is checked. Let E_1 be all ballots with valid proofs.
Creating $d_{i,j}$. The simulator $\mathcal{S}$ creates the $d_{i,j}$ by performing the necessary EPETs. $\mathcal{S}$ also creates the proofs honestly.
Converting $d_{i,j}$. $\mathcal{S}$ performs the protocol honestly and provides the proofs. During this process, it uses the secret key $SK_{\mathcal{T}}$ to decrypt the EPETs.
Creating o_i. $\mathcal{S}$ accumulates the $d_{i,j}$ as scheduled.
Creating final ballot list. For each ballot in E_1 create a cut list as described in the protocol. $\mathcal{S}$ creates a shuffle and the according proofs. Denote the result as E_2. $\mathcal{S}$ creates list of tuples, using the secret key: $E_3 := \{(\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i))\} : \mathsf{Pet}(o_i, \mathsf{Enc}(1)) = 1$
Checking Voter Credentials. $\mathcal{S}$ shuffles the second components of L_0 into a new list L_1: Let L_0' be the list of all encrypted public keys in L_0. $L_1 := \mathsf{Shuffle}(\mathsf{L}_0')$. For each ballot in E_3, $\mathcal{S}$ performs PETs with the second part of each entry. $\mathcal{S}$ uses the secret key for decryption. Let E_4 denote the list of all encrypted choices from the ballots with a according item in L_1.
Choice decryption. The decryption of the valid choices (E_4) is done by $\mathcal{S}$ with the secret key.
Adversarial guess. The adversary outputs a guess bit b'. $\mathcal{S}$ itself returns $d' := (b' \stackrel{?}{=} b)$ as his guess bit, i.e. whether the adversary correctly guessed b.

Since the simulator executes the protocol as specified, for $d = 1$ the simulation is indistinguishable to $\mathcal{A}$ from a real protocol. Let $\mathcal{V}$ denote the view of the adversary. Thus, we have

$$\Pr[S = 1 \mid d = 1] = Pr[\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}(\mathcal{V}) = 1] = \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}(\mathcal{V}).$$

On the other hand, as we argued above, the adversary in the simulation does not have any advantage to the adversary in $\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V})$ if $d = 0$. Thus,

$$\Pr[S = 1 \mid d = 0] = Pr[\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V}) = 1] = \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V}).$$

Concluding the argument, we have

$$\begin{aligned}\mathbf{Adv}_{\mathcal{S}}^{\mathsf{ddh}} &= \Pr[S = 1 \mid d = 1] - Pr[S = 1 \mid d = 0] \\ &= \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}(\mathcal{V}) - \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V}) \\ &= \mathbf{Adv}_{\mathcal{A}}^{revoting}.\end{aligned}$$

□

REFERENCES

Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. 2010. Signing on Elements in Bilinear Groups for Modular Protocol Design. Cryptology ePrint Archive, Report 2010/133. (2010). http://eprint.iacr.org/.

Masayuki Abe and Fumitaka Hoshino. 2001. Remarks on Mix-Network Based on Permutation Networks. In *Public Key Cryptography (Lecture Notes in Computer Science)*, Kwangjo Kim (Ed.), Vol. 1992. Springer, 317–324.

Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. 2009. Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios. In *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE'09)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1855491.1855501

Roberto Araújo, Sébastien Foulle, and Jacques Traoré. 2010. A Practical and Secure Coercion-Resistant Scheme for Internet Voting. In *Towards Trustworthy Elections*, David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida (Eds.). Lecture Notes in Computer Science, Vol. 6000. Springer Berlin Heidelberg, 330–342. DOI: http://dx.doi.org/10.1007/978-3-642-12980-3_20

Josh Benaloh. 2013. Rethinking Voter Coercion: The Realities Imposed by Technology. *Presented as part of the USENIX Journal of Election and Technology and Systems (JETS)* (2013), 82–87.

Jan Camenisch and Markus Stadler. 1997. Efficient group signature schemes for large groups. In *Advances in Cryptology CRYPTO '97*, Jr. Kaliski, BurtonS. (Ed.). Lecture Notes in Computer Science, Vol. 1294. Springer Berlin Heidelberg, 410–424. DOI: http://dx.doi.org/10.1007/BFb0052252

Ran Canetti and Rosario Gennaro. 1996. Incoercible Multiparty Computation. Cryptology ePrint Archive, Report 1996/001. (1996). http://eprint.iacr.org/.

David Chaum and Torben P. Pedersen. 1993. Wallet Databases with Observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '92)*. Springer-Verlag, London, UK, UK, 89–105. http://dl.acm.org/citation.cfm?id=646757.705670

Jeremy Clark and Urs Hengartner. 2011a. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Financial Cryptography (Lecture Notes in Computer Science)*, George Danezis (Ed.), Vol. 7035. Springer, 47–61.

Jeremy Clark and Urs Hengartner. 2011b. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. Cryptology ePrint Archive, Report 2011/166. (2011). http://eprint.iacr.org/.

Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A Secure and Optimally Efficient Multi-Authority Election Scheme. Springer-Verlag, 103–118.

Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2009. Verifying Privacy-type Properties of Electronic Voting Protocols. *Journal of Computer Security* 17, 4 (July 2009), 435–487. DOI: http://dx.doi.org/10.3233/JCS-2009-0340

Taher Elgamal. 1985. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology Proceedings of CRYPTO 84*, David Chaum George Robert Blakley (Ed.). Springer-Verlag, Berlin, Heidelberg, 10–18. http://www.springerlink.com/content/jl0mkpm32tn8ve3q/

Aleksander Essex, Jeremy Clark, and Urs Hengartner. 2012. Cobra: Toward Concurrent Ballot Authorization for Internet Voting. In *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE'12)*. USENIX Association, Berkeley, CA, USA, 3–3. http://dl.acm.org/citation.cfm?id=2372353.2372356

Kristian Gjøsteen. 2010. Analysis of an internet voting protocol. *IACR Cryptology ePrint Archive* 2010 (2010), 380.

Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. *SIAM J. Comput.* 17, 2 (April 1988), 281–308. DOI: http://dx.doi.org/10.1137/0217017

Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. 2004. Universal Re-encryption for Mixnets. In *Topics in Cryptology – CT-RSA 2004*, Tatsuaki Okamoto (Ed.). Lecture Notes in Computer Science, Vol. 2964. Springer Berlin Heidelberg, 163–178. DOI: http://dx.doi.org/10.1007/978-3-540-24660-2_14

James Green-Armytage. 2014. Direct Voting and Proxy Voting. (2014). Available at http://inside.bard.edu/~armytage/proxy.pdf.

Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. 2013. Caveat Coercitor: Coercion-Evidence in Electronic Voting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP '13)*. IEEE Computer Society, Washington, DC, USA, 367–381. DOI: http://dx.doi.org/10.1109/SP.2013.32

Jens Groth. 2002. A Verifiable Secret Shuffe of Homomorphic Encryptions. In *Public Key Cryptography - PKC 2003*, Yvo Desmedt (Ed.). Lecture Notes in Computer Science, Vol. 2567. Springer Berlin / Heidelberg, 145–160. http://dx.doi.org/10.1007/3-540-36288-6_11 10.1007/3-540-36288-6_11.

Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology (EUROCRYPT'08)*. Springer-Verlag, Berlin, Heidelberg, 415–432. http://dl.acm.org/citation.cfm?id=1788414.1788438

Markus Jakobsson and Ari Juels. 2000. Mix and Match: Secure Function Evaluation via Ciphertexts. In *Advances in Cryptology ASIACRYPT 2000*, Tatsuaki Okamoto (Ed.). Lecture Notes in Computer Science, Vol. 1976. Springer Berlin Heidelberg, 162–177. DOI: http://dx.doi.org/10.1007/3-540-44448-3_13

Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. ACM, New York, NY, USA, 61–70. http://markus-jakobsson.com/papers/jakobsson-wpes05.pdf

Ari Juels, Dario Catalano, and Markus Jakobsson. 2010. Coercion-Resistant Electronic Elections. In *Towards Trustworthy Elections*, David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida (Eds.). Lecture Notes in Computer Science, Vol. 6000. Springer Berlin Heidelberg, 37–63. DOI: http://dx.doi.org/10.1007/978-3-642-12980-3_2

Shahram Khazaei, Tal Moran, and Douglas Wikström. 2012. A Mix-Net from Any CCA2 Secure Cryptosystem. In *Advances in Cryptology ASIACRYPT 2012*, Xiaoyun Wang and Kazue Sako (Eds.). Lecture Notes in Computer Science, Vol. 7658. Springer Berlin Heidelberg, 607–625. DOI: http://dx.doi.org/10.1007/978-3-642-34961-4_37

Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2012. A Game-Based Definition of Coercion-Resistance and its Applications. *Journal of Computer Security (special issue of selected CSF 2010 papers)* 20, 6/2012 (2012), 709–764.

Mirosław Kutyłowski and Filip Zagórski. 2007. Verifiable Internet Voting Solving Secure Platform Problem. In *Advances in Information and Computer Security*, Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg (Eds.). Lecture Notes in Computer Science, Vol. 4752. Springer Berlin Heidelberg, 199–213. DOI: http://dx.doi.org/10.1007/978-3-540-75651-4_14

Epp Maaten. 2004. Towards Remote E-Voting: Estonian case.. In *Electronic Voting in Europe (LNI)*, Alexander Prosser and Robert Krimmer (Eds.), Vol. 47. GI, 83–100. http://dblp.uni-trier.de/db/conf/evoting/evoting2004.html#Maaten04

James C. III Miller. 1969. A program for direct and proxy voting in the legislative process. *Public Choice* 7, 1 (1969), 107–113. DOI: http://dx.doi.org/10.1007/BF01718736

Tal Moran and Moni Naor. 2006. Receipt-Free Universally-Verifiable Voting With Everlasting Privacy. In *Advances in Cryptology – CRYPTO 2006 (Lecture Notes in Computer Science)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, 373–392.

Andrew C. Myers, Michael Clarkson, and Stephen Chong. 2008. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy*. IEEE, 354–368. http://www.truststc.org/pubs/450.html

C. Andrew Neff. 2001. A verifiable secret shuffle and its application to e-voting. CCS '01 Proceedings of the 8th ACM conference on Computer and Communications Security. (2001).

Gerald V. Post. 2010. Using re-voting to reduce the threat of coercion in elections. In *Electronic Government, an International Journal (Volume 7, Number 2/2010)*. Inderscience Publishers, 168–182. https://inderscience.metapress.com/content/e841t68786434728/resource-secured/?target=fulltext.pdf

Riigikogu. 2002. Riigikogu Election Act. Riigi Teataja. (2002). https://www.riigiteataja.ee/en/eli/ee/501092014005/consolide/current.

Kazue Sako and Joe Kilian. 1995. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'95)*. Springer-Verlag, Berlin, Heidelberg, 393–403. http://dl.acm.org/citation.cfm?id=1755009.1755052

C.P. Schnorr. 1991. Efficient signature generation by smart cards. *Journal of Cryptology* 4, 3 (1991), 161–174. DOI: http://dx.doi.org/10.1007/BF00196725

Warren D. Smith. 2005. New cryptographic election protocol with best-known theoretical properties. Frontiers in Electronic Elections (FEE 2005). (2005).

Oliver Spycher, Rolf Haenni, and Eric Dubuis. 2010. Coercion-resistant hybrid voting systems. In *Electronic Voting*. 269–282.

Dominique Unruh and Jörn Müller-Quade. 2010. Universally Composable Incoercibility. In *Crypto 2010 (LNCS)*, Vol. 6223. Springer, 411–428. Preprint on IACR ePrint 2009/520.

Melanie Volkamer and Rüdiger Grimm. 2006. Multiple Casts in Online Voting: Analyzing Chances. In *Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria. (LNI)*, Robert Krimmer (Ed.), Vol. 86. GI, 97–106. http://subs.emis.de/LNI/Proceedings/Proceedings86/article4554.html

A. TAG CONVERSION

In Section 4.4 we informally describe how to create encrypted marks that tell which ballots have been superseded. Here we define the procedure more formally. The procedure starts with the list of all ballots and a corresponding list of differences next to each ballot and attaches an encrypted mark to each ballot. The encrypted mark tells whether the ballot has been superseded.

We use subroutines like a secret verifiable shuffle (see Section 4.1.1). Since each of those subroutines is verifiable, it produces a proof of correctness beside the actual result. Each proof contributes to the over-all proof of correctness. In abuse of notation and for better readability, we do not explicitly mention them.

Conversion $\mathsf{Convert}(L)$

Input. A list L with (ballot,differences)-entries $L[i] = (b_i, (d_{i,i+1}, \ldots, d_{i,n}))$

Output. A converted list L' of ballots and differences, and a list P of proofs of correctness.

Algorithm. (Each operation which can not be recomputed by everyone outputs a proof of correctness. For the sake of readability, we do not explicitly write the proof-output, but assume it implicitly appended to the list P.)

(1) Initialize list L' with $L[i] := (b_i, [])$ (the second entry will be filled later).
Initialize a look up table TS with an entry (i, ts_i) for each ballot $b_i = (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$

(2) For each ballot b_i prepare a tag $\mathsf{Enc}(\mathrm{ts}_i)$.

(3) $\forall 1 \le i \le n,\ \min(i+1, n) \le j \le n$: Save $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j}, [], [])$ to a list D. The empty entries will be filled later

(4) Shuffle D: $D' := \mathsf{Shuffle}(\mathsf{D})$.

(5) For each $d_{i,j}$ in D'
convert $d_{i,j}$ to $d'_{i,j} := \begin{cases} \mathsf{Enc}(r) & \text{if } d_{i,j} = \mathsf{Enc}(1) \\ \mathsf{Enc}(1) & \text{else} \end{cases}$ where $r \neq 1$
and store $d'_{i,j}$ in D' next to $d_{i,j}$: $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j}, d'_{i,j}, [])$.

(6) create the proof Π of correct conversion and store it in D': $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j}, d'_{i,j}, \Pi)$.
The correctness of the conversion can be verified based on D'.

(7) Shuffle D': $D'' := \mathsf{Shuffle}(\mathsf{D}')$

(8) For each $(\mathsf{Enc}(\mathrm{ts}_i), d'_{i,j}) \in D''$:
— Decrypt tag $\mathsf{Enc}(\mathrm{ts}_i)$ to ts_i with proof of correct decryption
— look up entry (i, ts_i) in TS
— append $d'_{i,j}$ to the second entry of $L'[i]$.

(9) For each ballot b_i: multiply each element $d'_{i,j}$ in the second entry of L'[i] and replace them by the result

(10) Output (L', P).

Fig. 10: Marking each ballot whether it has been superseded.

B. CORRECTNESS

Recall that a voting scheme is correct if the success probability of any adversary in Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$ (Figure 2) is negligible in $\max(k_1, k_2, k_3)$. We show that our construction is correct by an induction over the number of ballots n on the bulletin board in Step 8 of experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$. The number $n = n_{\mathcal{V}} + n_{\mathcal{A}}$ is comprised of $n_{\mathcal{V}}$, the number of ballots posted by honest voters on the bulletin board ($\mathcal{BB}$) in Step 4, and $n_{\mathcal{A}}$, the number of ballots posted on $\mathcal{BB}$ by the adversary in Step 6. Per assumption, the bulletin board is append-only.

Fix $n = 1$ as the base case, i.e. either $n_{\mathcal{A}} = 0$ or $n_{\mathcal{V}} = 0$. If $n_{\mathcal{A}} = 0$, then $\mathbf{X}' = \mathbf{X}$ and $P' = P$, and the experiment outputs 0. On the other hand, if $n_{\mathcal{V}} = 0$, then we have to show that the adversary can only cast a vote for a corrupted voter. (The adversary cannot overwrite an uncorrupted voter's vote, because there are none.) Ballots containing an invalid voter public key, i.e. a public key $pk \notin L_0$ are discarded in the tallying phase (see Section 4.4, Step 5). The probability that the adversary

generates a public key pair (sk, pk) such that sk generates a valid signature for the ballot and $pk \in L_0$ is negligible in the security parameter of the signature scheme. Thus, the experiment outputs 0 except for a negligible probability. (We stress that "malformed" ballots without a correct proof of knowledge of a signature on the ballot are discarded before the tallying phase of our protocol.)

Now assume there are $n = n_{\mathcal{V}} + n_{\mathcal{A}}$ ballots on the bulletin board in Step 8 of $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$ and the probability that the experiments outputs 1 is negligible in the security parameters for any adversary. To transfer to the $n+1$-case, we distinguish two cases to post an additional ballot on $\mathcal{BB}$.

— The additional ballot is cast in Step 4, i.e. by an uncontrolled voter. We have to show that it supersedes the most recent ballot by the same voter. This is achieved by a pairwise comparison of the public keys on the ballots (see Section 4.4).
— The additional ballot is cast in Step 6, i.e. by a controlled voter. We have to show that it does not supersede a ballot by a different voter and itself is not already superseded. By the same argument as above, only ballots that contain identical public keys are superseded during the tallying phase. (For the adversary to post a ballot containing the public key of an uncontrolled voter, he would have to forge the signature of the ballot and thus break the EUF-CMA security of the signature scheme.) Because the newest ballot has the newest timestamp per assumption, it is not superseded in this step.

This concludes the argument.

□

C. VERIFIABILITY

We argue the verifiability of our voting scheme informally. A process composed of several subroutines is verifiable if each step is verifiable by itself. Our process is composed of a small set of verifiable black-box functionalities and other verifiable buildings blocks. We describe them in Sections 4.1 and 4.2.

Each of these subroutines produces proofs of correctness along with its result. If either of these proofs is invalid, $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$ outputs 0. All these proofs are published next to the result of the subroutine on the bulletin board. The whole bulletin board with the input $\{b_i\}_{i=1}^{n}$, the final output $\mathbf{X}$, all subroutine proofs, and all interim results is the global proof P of correctness of the tally. We point out that each interim result is published except for the secret keys and some randomness which is necessary to protect the privacy of the voter. The input of each subroutine is also marked down on the bulletin board. Sometimes the output from one routine has to be reshaped to match the input format from another one. This reshaping is always deterministic.

Several operations like the reshaping or the computation of the product over all marks $d_{i,j}$ of each ballot b_i can be recomputed by anyone. Therefore no explicit proof is necessary for these steps. All operations which are not recomputable by the public are accompanied by proofs of correctness.

In conclusion, our tallying process is verifiable because intermediate values are public, and all subroutines are publicly verifiable.

USENIX Journal of Election Technology and Systems (JETS)
Volume 3, Number 2 • August 2015
www.usenix.org/jets/issues/0302

New Techniques for Electronic Voting

ALAN SZEPIENIEC, Dept. of Electrical Engineering — ESAT/COSIC, KU Leuven and iMinds, Belgium
BART PRENEEL, Dept. of Electrical Engineering — ESAT/COSIC, KU Leuven and iMinds, Belgium

This paper presents a novel unifying framework for electronic voting in the universal composability model that includes a property which is new to universal composability but well-known to voting systems: universal verifiability. Additionally, we propose three new techniques for secure electronic voting and prove their security and universal verifiability in the universal composability framework. 1. A tally-hiding voting system, in which the tally that is released consists of only the winner without the vote count. Our proposal builds on a novel solution to the millionaire problem which is of independent interest. 2. A self-tallying vote, in which the tally can be calculated by any observer as soon as the last vote has been cast — but before this happens, no information about the tally is leaked. 3. Authentication of voting credentials, which is a new approach for electronic voting systems based on anonymous credentials. In this approach, the vote authenticates the credential so that it cannot afterwards be used for any other purpose but to cast that vote. We propose a practical voting system that instantiates this high-level concept.

1. INTRODUCTION

For over two thousand years, voting systems have been in use. Recent developments in cryptography have enabled the transition from roll call votes or paper ballots to electronic voting systems. The application of cryptography to voting systems does not merely ensure correctness and security; on the contrary, it improves on these and offers new desirable properties when compared to its low-tech predecessors. However, the use of cryptography in this context leaves society no option but to trust a small number of cryptographers, and hardware and software vendors who develop and distribute these cryptographic voting systems.

Even more recently, the internet has revolutionized communication and the decision making process. What used to be a theoretical exercise is now becoming a practical reality: the internet allows us to hold votes and even state elections from our own personal computers.

However, the internet has had little effect on the theoretical foundation of electronic voting systems. The formalisms and techniques proposed in other research as well as in this paper make abstraction of the communication channel and apply equally to any voting system *not* conducted over the internet. On the other hand, there is one very important implication of internet voting that bears endless repeating: no mechanism can prevent the coercer from being physically present when the voter casts his vote, and hence *internet voting is inherently coercible*. In order to guard against coercion, a fundamentally different adversarial model must be considered. While this is addressed in the literature [Jonker and de Vink 2006] [Benaloh and Tuinstra 1994] [Sako and Kilian 1995] [Canetti and Gennaro 1996], coercive adversaries are beyond the scope of this paper.

The standard framework for assessing the security of cryptographic protocols is the universal composability (UC) framework proposed by Canetti [Canetti 2001]. While the UC framework imposes stringent constraints on the protocols and their design, it is beneficial for at least two reasons. First, the UC framework applies to *any* protocol and not just a predefined subset of them. Second, universal composability guarantees that the composition of UC-secure protocols is itself UC-secure, whether the components are invoked once or multiple times, recursively or not, serially or in parallel. This principle enables protocol designers to adopt a modular approach whereby the global protocol is composed of smaller subprotocols, each of which is proved UC-secure in its own right.

A *voting system* is a protocol like any other and hence the UC framework is applicable not just to the whole but also to the various subprotocols. Groth [Groth 2004b], for example, uses the UC framework to assess the security of some cryptographic voting systems. However, this assessment applies only to homomorphic aggregation voting systems and does not extend to mixnet-based or anonymous credential-based voting systems. Moreover, it does not take into account universal verifiability, *i.e.* the ability of any observer to verify the tally. Other formalisms, not explicitly taking UC-security into account, include the ones by Benaloh [Benaloh 1987], Cramer *et al.* [Cramer et al. 1996; Cramer et al. 1997], Adida [Adida 2006] and Chevallier-Mames [Chevallier-Mames et al. 2010].

Our contributions. We present a new formalism of voting systems which is more general than that of Groth as it contains homomorphic aggregation schemes as a special case. We also take the properties introduced in the other formalisms into account. Additionally, we formalize and prove a composability theorem for universal verifiability. We note that, at first sight, this notion of universal verifiability seems similar to the notion of public auditability for multiparty computation protocols recently and independently introduced by Baum, Damgård and Orlandi [Baum et al. 2014]. However, there are several important differences, which will be explained in due course.

Moreover, we introduce three new techniques for electronic voting protocols and subsequently prove their UC-security and, hence, their compatibility with our formalism. First, we propose a "Tally-Hiding Vote": a voting system where the result of the tallying process identifies the winner of the vote but leaks no information on the vote count. Our proposal uses a novel solution to Yao's millionaire problem [Yao 1982] which depends on a well-known property of Damgård and Jurik's threshold cryptosystem [Damgård and Jurik 2001], namely the ability to lift a ciphertext from one ciphertext space into another, larger space without modifying the plaintext. Our solution to the millionaire problem invokes this lifting procedure as an abstract black box. While Damgård and Jurik have proposed one instantiation of this protocol [Damgård and Jurik 2002], we propose another one which is tailored to our particular use case.

Second, we propose a new "Self-Tallying" voting system, along the lines of Kiayias–Yung [Kiayias and Yung 2002] and Groth [Groth 2004a]. In this type of voting system, the computationally expensive part of the tallying process is completed beforehand. The tally is known — or is easily computed by anyone — as soon as the last voter has cast his or her vote but not before. The obvious drawback is that one voter can boycott the entire procedure by refusing to cast his or her vote. The essence of our approach, like that of Kiayias and Yung, is that the randomizers used by the voters to encrypt their votes are not entirely random but in fact cancel out when aggregated. As long as at least one voting authority is not corrupt, the authorities are unable to determine any one voter's particular random value or vote. Our system allows for a practically unlimited number of secure votes after just one initialization procedure of constant size, in contrast to the scheme of Kiayias and Yung where this initialization procedure is linear in the number of intended elections.

Third, we present a new paradigm for credential-based voting, namely "Authenticated Voting Credentials", along with a scheme that implements this concept. In contrast to previous schemes for electronic voting based on anonymous credentials, such as those of Chaum [Chaum 1988], Fujioka *et al.* [Fujioka et al. 1992], and Ohkubo *et al.* [Ohkubo et al. 1999], our credentials are authenticated by the vote that is cast. In this setting, a man-in-the-middle cannot intercept a credential and append it to his own vote or just modify the vote; at most, he can prevent the voter's vote from being cast by blocking it entirely. While the concept applies to *any* credential system, we demonstrate it by applying it to the credential system by Ferguson [Ferguson 1993], which draws from Chaumian blind signatures [Chaum 1982], and which we use in combination with Guillou and Quisquater's zero-knowledge proof of an RSA signature [Guillou and Quisquater 1988].

Outline. We start by summarizing universal composability in Section 2 after which we present our formalism of electronic voting and universal verifiability. In the subsequent sections, we cover our three new techniques: Tally-Hiding Vote in Section 3, Self-Tallying Vote in Section 4, and Authenticated Voting Credentials in Section 5. Finally, Section 6 concludes the text.

Notation and Preliminaries. Let $\kappa \in \mathbb{N}$ be a security parameter. We say a function $\varepsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ of the security parameter is *negligible* if $\forall c \in \mathbb{R}_{>0} \,.\, \exists K \in \mathbb{N} \,.\, \forall \kappa \geq K \,.\, \varepsilon(\kappa) \leq \kappa^{-c}$. A *public-key encryption scheme* consists of a tuple of possibly probabilistic algorithms $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ where G generates a matching public and private key; E is an algorithm that encrypts a given message using the public key and D decrypts a ciphertext provided that the matching private key is used. A public key cryptosystem is *semantically secure* if no non-trivial function of the plaintext can be computed given only the ciphertext and the public key. By $x \leftarrow y$ we denote the assignment of the value y to the

variable x; by $x \xleftarrow{\$} S$ we denote a uniformly random selection from a finite set S and assignment of this value to the variable x. Long arrows with values on top denote messages sent between parties. For any positive integer $n \in \mathbb{N}$, we use $\mathbb{Z}_n$ to denote, depending on the context, either the ring of integers modulo n, or the set of integers $\{0,1,\ldots,n-1\}$. Likewise, we use $\mathbb{Z}_n^*$ to denote either the group of integers under multiplication modulo n or else the subset of $\mathbb{Z}_n$ containing integers that are invertible for multiplication modulo n.

The Paillier cryptosystem is a semantically secure public-key scheme whereby the public key is given by n, a product of two large private primes, and the secret key is given by d, a number which satisfies $d = 0 \mod \varphi(n)$ and $d = 1 \mod n$. Encryption of a message m using randomness r is defined as $\mathsf{E}(m) = (1+n)^m r^n \mod n^2$. Decryption consists of first raising the ciphertext to the exponent d in order to cancel the randomizer, and secondly computing the easy discrete logarithm of the resulting value.

2. SECURE ELECTRONIC VOTING

2.1. Universal Composibility

The key idea of universal composability [Canetti 2001] is the comparison between two worlds in both of which a set of participants compute a function of their inputs. In one world, they hand their inputs to a trusted third party called the *ideal functionality* which calculates the result and hands the correct outputs to the respective parties. In the other world, the computation is performed by a protocol executed by the parties. In both worlds, there is an external environment $\mathcal{E}$, modeled as a polynomial-time program or Turing machine, which chooses the inputs of the participants and reads their outputs. The environment interacts with the adversary throughout the process.

A protocol is a *UC-secure realization* of the ideal functionality if for every adversary $\mathcal{A}$ interacting with the participants while they execute the protocol, there exists a simulator $\mathcal{S}$ which interacts with the ideal functionality and with the parties involved such that no external environment $\mathcal{E}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ or with $\mathcal{S}$. This implies that any attack that can be performed by $\mathcal{A}$ against the protocol can also be performed by $\mathcal{S}$ against the ideal functionality.

The power of the UC framework is demonstrated by the universal composability theorem: if $\mathcal{P}_1$ securely realizes an ideal functionality $\mathcal{F}$, and if a protocol $\mathcal{P}_2$ in which $\mathcal{F}$ is used as subprotocol is secure, then replacing $\mathcal{F}$ by $\mathcal{P}_1$ will not change the security of $\mathcal{P}_2$. This theorem allows us to design secure protocols by designing secure modules which realize smaller functionalities.

2.2. Voting

An *option* is any string of characters that is known by the voters to refer to a hypothetical situation. Let $\mathbf{O}$ denote the set of options for a given vote; and let $\mathsf{P}\mathbf{O}$ denote the set of permutations of $\mathbf{O}$, which corresponds to the set of expressible preferences with respect to $\mathbf{O}$.[1] A *vote* (in the first sense of the word) is an expressed preference with respect to the options and thus an element of $\mathsf{P}\mathbf{O}$.

A *tally* is any string of characters which directly implies a partial ordering on the set of options — an ordering where some subsets are allowed to have the same rank. A tally may be the encoding of a mapping of the options to their vote counts, or of a string which identifies only the winning option or of something else. Let $\mathbf{T}$ denote the set of tallies.

A *tallying function* f is any function that maps n votes to a tally:

$$f : (\mathsf{P}\mathbf{O})^n \to \mathbf{T}.$$

A *voting system* is an interactive protocol which evaluates a tallying function in the set of votes cast by the voters. We distinguish two classes of participants: voters $V_1, V_2, \ldots, V_n \in \mathbf{V}$, whose votes should be counted by the tallying function; and authorities $A_1, A_2, \ldots, A_k \in \mathbf{A}$, who bear the respon-

[1] Preferences are always necessarily ordinal. This implicit ordering is manifested when the ballot does not allow "no preference". This implicit ordering may be not encoded if the ballot does allow it.

sibility for the correct execution of the protocol and who may be expected to perform more work. The protocols are designed to work as long as a subset of size t of the authorities are not corrupt.

The second meaning of the word "vote" refers to one instantiation or execution of a voting system. In an electronic voting setting, all participants are assumed to have access to a public append-only bulletin board. In the Section 2.3, we model this bulletin board by an ideal functionality.

We aim to design voting systems which are *universally verifiable*, meaning that *anyone* can verify the correctness of the protocol execution after its execution by requesting a transcript which certifies this. Provided that verification succeeds, universal verifiability guarantees *correctness*, which means that the tally that results from the protocol conforms to the tallying function and the cast votes. Correctness, in turn, implies *completeness* — all valid votes are counted correctly; *soundness* — invalid votes are not counted; *eligibility* — only votes from eligible voters $V_i \in \mathbf{V}$ are counted; *unreusability* — no voter can vote in excess of his or her quota (possibly but not necessarily one vote per person); and *finality* — no voter can change his or her vote after the deadline has expired.

In addition to correctness, we want the voting system to ensure privacy. This feature is formalized by *perfect ballot secrecy*: the adversary cannot compute any information on any one individual's vote beyond what follows from the tally (and from the votes of the corrupted parties). Perfect ballot secrecy implies *fairness*: no partial tallies can be known before the deadline — or, for that matter, at any other point in time.

Other desirable features are *receipt-freeness*, where a voter cannot prove that he voted one way or another and, related to it, *uncoercibility*, where a voter cannot be coerced into voting a specific way even by a coercer who interacts with them *while casting the vote*. These properties fall outside of the scope of this paper.

2.3. UC-Secure Voting Systems

A voting system can only be UC-secure if it is a secure realization of an *ideal functionality*. Hence, we define the ideal functionality to be realized as well as the ideal functionalities that are allowed to be invoked. We start with the second, from the bottom upwards.

Groth [Groth 2004b] correctly notes that the bulletin board as well as a public key generation mechanism are in fact ideal functionalities. While Groth presents the two functionalities as a single ideal functionality, we opt to separate the two. The reason is that one might want to use the one functionality without using the other. The bulletin board is formalized by the ideal functionality $\mathcal{F}_{\mathsf{BB}}$ and the system public key generation mechanism is formalized by the ideal functionality $\mathcal{F}_{\mathsf{SKG}}$. The functionalities run with voters $V_1, \ldots, V_n$ and authorities $A_1, \ldots, A_k$, and with either an adversary $\mathcal{A}$ or adversary-simulator $\mathcal{S}$ (but for the purpose of describing the ideal functionalities, these two are interchangeable).

The bulletin board is a public append-only database of messages (any string of characters) from any protocol participant. We extend the power of the adversary compared to Groth. In addition to blocking honest voters' messages, the adversary in our bulletin board model can even falsify their messages — and the same holds for the messages of authorities. This is accomplished as follows: the functionality forwards all messages not originating from the adversary, to the adversary; the adversary can choose to allow them to the bulletin board by instructing the ideal functionality to do so. In contrast, the adversary cannot prevent voters nor authorities from reading bulletin board messages. Access to the bulletin board is not anonymous as all messages pass through the adversary. However, in Section 5 we deviate from this model as we are only able to prove UC-security of a credential-based voting system on the premise that access to the bulletin board *is* anonymous, *i.e.* , the adversary cannot determine who is posting messages to the bulletin board or even whether this communication is taking place. The bulletin board functionality $\mathcal{F}_{\mathsf{BB}}$ is presented in Figure 1.

The system key generation mechanism is straightforward. The functionality starts by generating a public key, matching private key shares and verification keys for those shares. The private key shares are distributed to the proper authorities. The public key as well as the verification keys are distributed to everyone such that anyone who assumes the role of the verifier can verify the correct execution of the authorities' distributed decryption protocol. The functionality is parameterized by

Functionality $\mathcal{F}_{\mathsf{BB}}$

— Upon receiving $(\textbf{message}, sid, m)$ from any participant $P_i \in \mathbf{V} \cup \mathbf{A}$, send $(\textbf{message}, sid, P_i, m)$ to $\mathcal{S}$.
— Upon receiving $(\textbf{post}, sid, m)$ from $\mathcal{S}$, store it.
— Upon receiving $(\textbf{read}, sid)$ from any participant $P_i \in \mathbf{V} \cup \mathbf{A} \cup \{\mathcal{S}\}$, send to P_i the list of all stored tuples $(\textbf{post}, sid, *)$ in chronological order.

Fig. 1: bulletin board.

the public key cryptosystem to instantiate. For example, in a homomorphic aggregation type voting system, an additively homomorphic public key encryption scheme is required while for a credential-based voting system, a blind signature scheme is necessary. The system key generation functionality is presented in Figure 2.

Functionality $\mathcal{F}_{\mathsf{SKG}}$

— Upon invocation, generate keys for a public key threshold cryptosystem. Send $(\textbf{public key}, sid, pk)$, where pk is the public key, to all participants and to $\mathcal{S}$. Send to all participants and to $\mathcal{S}$ the tuple $(\textbf{verification keys}, sid, vk_1, \ldots, vk_k)$ where vk_j is the verification key for A_j's key share. Send to all authorities $A_j \in \mathbf{A}$ the message $(\textbf{secret share}, sid, sk_j)$ where sk_j is the secret key share for A_j.

Fig. 2: system key generation mechanism.

While more realistic than the alternative, allowing the adversary to also falsify voters' messages does come with a price. The authorities need a mechanism by which to separate eligible voters from non-eligible voters (or more precisely: separate messages from eligible voters from messages from an adversary masquerading as eligible voter). We achieve this property by introducing an ideal participant key generation functionality, $\mathcal{F}_{\mathsf{PKG}}$. The ideal participant key generation functionality generates two public and private keys for each participant — one pair for encryption and one pair for signatures. The private keys are communicated to the respective participants whereas the public keys are sent to all participants including the adversary. After invoking the participant key generation functionality, participants can authenticate all their outward communication if they should choose to do so by signing it with their private signature key. After this functionality invocation, anyone can distinguish between eligible and non-eligible voters based on the list of eligible voter public keys.

The ideal participant key generation functionality also turns out to be of crucial importance for universal verifiability. We will formally define the concept later but at this point we introduce a verifier $\mathcal{V}$, which is another party in the protocol. The public keys produced by $\mathcal{F}_{\mathsf{PKG}}$ are also sent to the verifier, who uses them to verify that genuine interaction between real parties has taken place. Since the adversary does not possess the private keys from honest parties, he cannot fake their communications. The verifier, too, can distinguish between votes from eligible voters and votes from non-eligible voters based on the list of eligible voter public keys. $\mathcal{F}_{\mathsf{PKG}}$ is formally presented in Figure 3.

Having properly formalized the ideal functionalities which a voting system might invoke, we can turn to the voting system itself. The ideal voting system, $\mathcal{F}_{\mathsf{VS}}$, is presented in Figure 4. Here, in contrast to the bulletin board functionality, the adversary is not allowed to read the cast votes or to falsify them. However, he *is* allowed to block them. In fact, the ideal functionality blocks the votes by default and requires the intervention of the adversary, in the form of a **no-block** message, to unblock the votes.

A UC-secure realization of $\mathcal{F}_{\mathsf{VS}}$ must necessarily satisfy the correctness and perfect ballot secrecy properties. Thus, if we prove that a particular voting system is a UC-secure realization of $\mathcal{F}_{\mathsf{VS}}$, then we prove that all the implied properties enumerated at the end of Section 2.2 are satisfied (i.e.:

Functionality $\mathcal{F}_{\mathsf{PKG}}$

— Upon invocation, for each voter $V_i \in \mathbf{V}$, generate a public/private key pair for encryption and a public/private key pair for signatures. Send the private keys to the V_i and the public keys to all participants including $\mathcal{S}$ and the verifier $\mathcal{V}$.
— Upon invocation, for each authority $A_j \in \mathbf{A}$, generate a public/private key pair for encryption and a public/private key pair for signatures. Send the private keys to the A_j and the public keys to all participants including $\mathcal{S}$ and the verifier $\mathcal{V}$.

Fig. 3: eligible participant key generation mechanism.

Functionality $\mathcal{F}_{\mathsf{VS}}$

— Upon receiving $(\textbf{vote}, sid, v_i)$ from voter $V_i \in \mathbf{V}$ and if v_i is a valid vote, store $(\textbf{vote}, sid, v_i, V_i)$ and send $(\textbf{vote}, sid, V_i)$ to $\mathcal{S}$. (Note that the vote v_i itself is *not* sent to the adversary.)
— Upon receiving $(\textbf{no-block}, sid, V_i)$ from $\mathcal{S}$, store it for future reference.
— Upon receiving $(\textbf{tally}, sid)$ from all the authorities, compute the tallying function on all the stored votes v_i from unique voters and for a matching **no-block** message was stored. If a voter V_i has multiple stored votes in excess of his quota, choose the most recent quota-sized subset of votes for which there is a matching **no-block** message. Calculate $t = f(v_1, \ldots, v_n)$ and send $(\textbf{result}, sid, t)$ to all parties including $\mathcal{S}$ and halt.

Fig. 4: voting system.

correctness — completeness, soundness, eligibility, unreusability, and finality — as well as perfect ballot secrecy). However, universal verifiability is less straightforward.

A naïve approach to universal verifiability might grant the verifier access to the bulletin board and demand that the transcript be of such a form that the verifier can easily catch any cheater. This approach is insufficient because it does not adequately capture the power of the adversary who attacks the protocol and who might be spoofing the verifier's read access to the bulletin board or manipulate its contents after the protocol has been executed.

However, we can demand that there be a transcript — either because a correct execution of the protocol produced it, or else because the adversary is trying to fool someone with it. We can then formalize universal verifiability as follows.

Definition 2.1. universal verifiability Let $\mathcal{V}$ (verifier) be a probabilistic polynomial-time algorithm which takes as input the transcript T as produced by an adversary $\mathcal{A}$ attacking a protocol $\mathcal{P}$. $\mathcal{V}$ eventually outputs a bit $\tilde{b}$. Let b be a variable indicating whether the protocol was executed correctly by all parties — *i.e.*, the parties behaved honestly and were not corrupted by the adversary — and if the transcript is authentic, by assuming the value 1 if this is the case and 0 otherwise. Then the protocol $\mathcal{P}$ is *universally verifiable* if there exists a verifier $\mathcal{V}$ such that, for all adversaries $\mathcal{A}$ attacking the protocol, $\mathcal{V}$ has significant distinguishing power:

$$\left|\Pr\left[b = \tilde{b}\right] - \Pr\left[b \neq \tilde{b}\right]\right| \geq \frac{1}{2} \ ,$$

where the probabilities are taken over all random coins used by $\mathcal{V}$, $\mathcal{A}$ and $\mathcal{P}$.

In other words, a protocol is universally verifiable if no adversary attacking it can convince the verifier that it was executed correctly by all parties when it was not. Without loss of generality, we assume that the verifier $\mathcal{V}$ indicates the transcript is authentic and the protocol execution was correct by outputting $\tilde{b} = 1$ and indicates the opposite with $\tilde{b} = 0$. The constant $\frac{1}{2}$ captures the notion of "significant" distinguishing power but in effect, any constant not negligibly close to 0 or 1 will do.

The verifier $\mathcal{V}$ trusts ideal functionalities and hence does not need to verify their correct execution. This makes the composability theorem for universal verifiability rather trivial.

THEOREM 2.2. ***If*** *(1) $\mathcal{P}_1$ is a hybrid protocol invoking the ideal functionality $\mathcal{F}_2$, and (2) $\mathcal{P}_1$ is universally verifiable for verifiers that trust $\mathcal{F}_2$, and (3) $\mathcal{P}_2$ is a universally verifiable protocol that realizes $\mathcal{F}_2$;* ***then*** *$\mathcal{P}_1$ is still universally verifiable if it invokes $\mathcal{P}_2$ rather than $\mathcal{F}_2$.*

PROOF. If there is a protocol $\mathcal{P}_1$ which invokes a universally verifiable ideal functionality $\mathcal{F}_2$, then the verifier $\mathcal{V}_1$ for $\mathcal{P}_1$ will not need to verify $\mathcal{F}_2$ as this functionality is trusted anyway. If $\mathcal{F}_2$ is realized by a universally verifiable protocol $\mathcal{P}_2$, then there must exist a verifier $\mathcal{V}_2$ that verifies its correct execution given a transcript T_2. Rather than trusting $\mathcal{F}_2$, $\mathcal{V}_1$ now requests a transcript T_2 for $\mathcal{P}_2$ from the adversary and simulates $\mathcal{V}_2$. $\mathcal{V}_1$ returns 0 if $\mathcal{V}_2$ returns 0. This is repeated for every invocation of $\mathcal{P}_2$. □

We stress that the verifier $\mathcal{V}$ must be able to differentiate between simulated parties created by the adversary $\mathcal{A}$ and genuine protocol participants. As such, the verifier must be a recipient of the public keys from $\mathcal{F}_{\mathsf{PKG}}$.

This notion of universal verifiability is in essence very similar to that of Baum, Damgård and Orlandi [Baum et al. 2014], which they name *public auditability*. Rather than in terms of a game between the verifier and the adversary, Baum *et al.* define public auditability in terms of the root ideal functionality which is extended so that it responds honestly to any participant who requests an audit. This response indicates whether or not parties in the protocol have behaved corruptly. This extension to the ideal functionality is realized by allowing anyone to have access to the bulletin board and view the transcript of the protocol. Anyone who wishes to audit, can retrieve a transcript of the protocol and decide themselves whether or not all participants behaved correctly. Only if the protocol is well-formed (for example, because the correctness of every step is proven by non-interactive zero-knowledge proofs), does the auditor stand a chance at deciding correctly, and only then can they be said to realize this extension to the ideal functionality.

Public auditability is dependent on the existence of the bulletin board, which must contain the transcript of the protocol. On the other hand, universal verifiability takes into account that the verifier's access to the bulletin board might be spoofed by the adversary. Instead, rather than relying on the existence of an append-only (even for the adversary) bulletin board, the verifier in the universal verifiability game requests the protocol transcript from the adversary. As such, universal verifiability is a stronger concept as it provides a similar notion of verifiable correctness in the face of a more powerful adversary.

Moreover, the verifier in the universal verifiability framework is able to verify the authenticity of participant's messages if they are accompanied by signatures, as the verifier has received the public keys from $\mathcal{F}_{\mathsf{PKG}}$. On the other hand, the auditor in the public auditability framework trusts the bulletin board to record the identities of whoever posts a message — but if the bulletin board guarantees this, then it precludes anonymous access. Universal verifiability does not depend on the existence of a bulletin board and does not impose such stringent constraints. Hence, a protocol based on anonymous communication cannot be publicly auditable, while it can be universally verifiable — possibly by employing anonymous credentials.

Lastly, universally verifiable protocols are composable, and hence integrate nicely with universally composable protocols. On the other hand, public auditability was designed specifically for multiparty computation and does not explicitly take into account modular design of just any protocol in the same way the UC framework does.

3. TALLY-HIDING VOTE

If the goal is to preserve voter privacy, the logical extreme is to release only the winning option of the vote and not the number of votes each option received. This idea to "show the winner without the tally" was first introduced for electronic voting by Benaloh [Cohen 1986]. We present a relatively efficient instantiation of this idea.

The key primitive used is a novel solution to the millionaire problem. Given two Paillier ciphertexts where the secret key is distributed among the voting authorities, there exists an efficient protocol to determine which of the two plaintexts is larger. For now, we consider only the case where there are two options. This strategy is sufficient for computing the winner in a preferential election while keeping the particular vote counts secret, but this does leak the entire ranking of all the options. Later on, we show a solution to calculate the winner among r options all the while leaking no information either on the ranking of the other options or on the particular vote counts.

Our scheme builds on a well-known property of the Damgård-Jurik cryptosystem [Damgård and Jurik 2001] (a generalization of Paillier's cryptosystem [Paillier 1999]) which allows the holders of the secret key to turn a ciphertext from one space, $\mathbb{Z}^*_{n^2}$ into a ciphertext from another space, $\mathbb{Z}^*_{n^3}$ such that it encrypts the same plaintext. Under certain conditions on the plaintext no information is leaked by this procedure (except that the plaintext satisfies said conditions). We invoke this *ciphertext lifting* procedure as a black box. While Damgård and Jurik themselves were the first to propose an instantiation of this protocol, we propose an alternate one which is tailored to our needs.

First, we prove that, given access to an ideal functionality $\mathcal{F}_{\mathsf{MP}}$ which solves the millionaire problem for two ciphertexts, we can construct a UC-secure tally-hiding voting system. Then, we show how we can realize $\mathcal{F}_{\mathsf{MP}}$ using ciphertext lifting as a black box ideal functionality. In Appendix A we present our alternative to Damgård and Jurik's lifting procedure.

3.1. UC-Secure Tally-Hiding Vote

In the original millionaire problem [Yao 1982], two millionaires want to decide who of them is richer without disclosing any other information about their wealth. In our case, there are k participants, each holding a share in the private key, who want to decide which one of two ciphertexts encrypts the larger plaintext without revealing any other information. For now, we assume we have a solution to this problem in the form of an ideal functionality and base a voting system on it. Later on, we present such a solution.

We define the ideal functionality $\mathcal{F}_{\mathsf{MP}}$ for the millionaire problem in Figure 5. The functionality takes two ciphertexts c_1, c_2 as input and compares the corresponding plaintexts, i.e.: it outputs 1 if $\mathsf{D}(c_1) < \mathsf{D}(c_2)$ and 0 otherwise. The functionality runs with all authorities. This functionality may be considered part of $\mathcal{F}_{\mathsf{SKG}}$, the system key generation functionality.

Functionality $\mathcal{F}_{\mathsf{MP}}$

— Upon receiving $(\mathbf{mp}, c_1, c_2, sid)$ from all authorities A_j, first check if all messages are identical. If not, do nothing; otherwise respond as follows. If $\mathsf{D}(c_1) < \mathsf{D}(c_2)$, send $(\mathbf{smaller}, sid, c_1, c_2)$ to all authorities and $\mathcal{S}$. Otherwise, send $(\mathbf{not\ smaller}, sid, c_1, c_2)$ to all authorities and $\mathcal{S}$.

Fig. 5: millionaire problem solver.

Using the ideal functionalities $\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}}$ and $\mathcal{F}_{\mathsf{MP}}$, we can define a protocol $\mathcal{P}_{\mathsf{THVS}}$ which securely realizes $\mathcal{F}_{\mathsf{VS}}$ with a tally-hiding tallying function. This protocol is presented in Figure 6. The protocol runs with authorities and voters. There are two options: "A" and "B". (See Section 3.3 for a generalization with more than two options.) Each voter publishes two ciphertexts, one corresponding to each option. In addition, the voter supplies a non-interactive zero-knowledge proof that one of his ciphertexts is an encryption of 1 and the other is an encryption of 0. Such a proof can be constructed by applying the subset proof technique from Cramer *et al.* [Cramer et al. 1994] to a proof of plaintext knowledge which in the case of Paillier encryption amounts to a Shoup proof [Shoup 2000]. The authorities apply homomorphic aggregation to obtain two ciphertexts encoding the number of votes for each option. They subsequently invoke $\mathcal{F}_{\mathsf{MP}}$ twice to determine which option has received more votes.

Protocol $\mathcal{P}_{\mathsf{THVS}}$

— Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
— All voters $V_i \in \mathbf{V}$ use the system public key to encrypt their vote $v_i = (v_i^{(A)}, v_i^{(B)})$ and generate a proof that the encryption is the encryption of a valid vote. The encryption and the proof are signed and then sent to $\mathcal{F}_{\mathsf{BB}}$. (The adversary $\mathcal{A}$ is assumed to allow them to pass.)
— The authorities $A_j \in \mathbf{A}$ use the cryptosystem's homomorphic property to calculate the aggregates: encryptions c_1 and c_2 of the number of votes for "A" and for "B" respectively.
— The authorities invoke $\mathcal{F}_{\mathsf{MP}}$ twice. Once on input $(\mathbf{mp}, c_1, c_2, sid)$ and once on input $(\mathbf{mp}, c_2, c_1, sid)$.
— If $\mathcal{F}_{\mathsf{MP}}$ returns **smaller**, **not smaller**, all authorities send $(\mathbf{message}, sid, (A < B, sig))$ to $\mathcal{F}_{\mathsf{BB}}$. If $\mathcal{F}_{\mathsf{MP}}$ returns **not smaller**, **smaller**, all authorities send $(\mathbf{message}, sid, (A > B, sig))$ to $\mathcal{F}_{\mathsf{BB}}$. If $\mathcal{F}_{\mathsf{MP}}$ returns **not smaller**, **not smaller**, then all authorities send $(\mathbf{message}, sid, (A = B, sig))$ to $\mathcal{F}_{\mathsf{BB}}$. The adversary is assumed to allow all messages to pass. In these messages, *sig* represents a signature by the authority who sends the message on the tally.
— All voters V_i send $(\mathbf{read}, sid)$ to $\mathcal{F}_{\mathsf{BB}}$.

Fig. 6: tally-hiding voting system.

$\mathcal{P}_{\mathsf{THVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$. We demonstrate this by showing that for every adversary $\mathcal{A}$ attacking an execution of the protocol, there exists an adversary-simulator $\mathcal{S}$ attacking the ideal process such that a computationally bounded external environment machine $\mathcal{E}$ cannot determine whether he is interacting with $\mathcal{A}$ and an execution of $\mathcal{P}_{\mathsf{THVS}}$ or with $\mathcal{S}$ and an execution of $\mathcal{F}_{\mathsf{VS}}$. Moreover, we must show that there exists a verifier $\mathcal{V}$ who queries the adversary for a transcript of $\mathcal{P}_{\mathsf{THVS}}$ and who returns 1 with high probability if and only if the protocol was executed correctly and 0 otherwise.

THEOREM 3.1. *Protocol $\mathcal{P}_{\mathsf{THVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ with a tally-hiding tally function in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}})$-hybrid model for non-adaptive adversaries corrupting up to $k-t$ authorities, where k is the number of authorities and t is the threshold of honest authorities.*

The proof is in Appendix B.1

3.2. Ciphertext Lifting and the Millionaire Problem

We now describe a secure protocol for solving the millionaire problem. Let us first summarize threshold decryption of a Paillier or Damgård-Jurik ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$. Here, n is the product of two large primes and s is a small integer (equal to 1 for the Paillier case).

The private decryption exponent, d, is distributed among k parties such that at least t of them must cooperate in order to perform a decryption. Every participant A_i (indexing starts from 1) is in possession of a value s_i, such that (i, s_i) is a point on the $(t-1)$-degree polynomial $f(x) \in \mathbb{Z}[x]$. This polynomial evaluates in zero to the secret exponent: $f(0) = d$ where d satisfies $d = 0 \mod \varphi(n)$ and $d = 1 \mod n$

In addition to a share s_i in the secret exponent, all parties have a public verification key $w_i = w^{\Delta s_i} \mod n^s$ for some fixed w. The verification key binds them to their secret share. Here, $\Delta = k!$.

In order to decrypt a ciphertext $c \in \mathbb{Z}^*_{n^s}$, the parties proceed as follows. Each party publishes a decryption share $c_i = c^{2\Delta s_i} \mod n^s$ along with a non-interactive zero-knowledge proof that $\mathrm{dlog}_{c^2} c_i = \mathrm{dlog}_w w_i$.

Once we have a set S of at least t decryption shares, we can combine them using Lagrange interpolation:

$$c' = \prod_{i \in S} c_i^{2\lambda_i} \mod n^s \quad \text{where } \lambda_i = \Delta \prod_{j \in S \setminus \{i\}} \frac{-j}{i-j} \ .$$

After combining the shares, the randomizer in the ciphertext will have been canceled and the resulting value will be of the form $c' = (1+n)^{4\Delta^2 m} \mod n^s$. Damgård and Jurik describe an algorithm to efficiently compute the plaintext $m \in \mathbb{Z}^*_{n^{s-1}}$ from this value.

Our lifting procedure is founded on the following observation. As soon as the authorities decide on an access structure Λ (a set of size t of indices of the authorities that will proceed with the protocol), these authorities are able to obtain a multiplicative sharing of the plaintext modulo n^2. Each authority A_i computes $c_i = c^{4\Delta^2 \prod_{j \in \Lambda \setminus \{i\}} \frac{-j}{i-j}} \mod n^2$ and then $(1+n)^m = \prod_{i \in \Lambda} c_i \mod n^2$ is guaranteed to hold. A little more juggling results in an additive sharing modulo n where $m = \sum_{i \in \Lambda} u_i \mod n$ is guaranteed to hold. Under favorable conditions on the plaintext, we can find an equivalent additive sharing where no overflow occurs: $m = r + \sum_{i \in \Lambda} u'_i$ (where u'_i remain secret). This sum is subsequently computed homomorphically in ciphertext-domain by first encrypting r and all the shares under the Damgård-Jurik cryptosystem ($s = 2$) and then multiplying all the summands. The protocol is described in full detail in Appendix A.

By contrast, Damgård and Jurik's solution [Damgård and Jurik 2002] involves regarding a ciphertext $c \in \mathbb{Z}^*_{n^s}$ as though it were a ciphertext in $\mathbb{Z}^*_{n^{s+1}}$: $c = \mathsf{E}_s(m,r) = \mathsf{E}_{s+1}(m + tn^s, r')$. The protocol proceeds to compute t from this value — by first homomorphically adding sufficiently small noise R, then decrypting and lastly by performing the division by n^s on $m + R + tn^s$. The desired ciphertext is obtained by encrypting tn^s and homomorphically calculating $\mathsf{E}_{s+1}(m + tn^s - tn^s, r'')$. However, this solution is cumbersome as it requires expensive range proofs on m and R in order to guarantee on the one hand that no overflow will occur and on the other hand that no information on m is leaked.

In order to use this to solve the millionaire problem, we make the following observation. For the Paillier cryptosystem, multiplication of two ciphertexts modulo n^2 homomorphically corresponds to addition of the plaintexts modulo n. However, for the Damgård-Jurik extension, multiplication of two ciphertexts modulo n^3 homomorphically corresponds to addition of the plaintexts modulo n^2. But modulo n^2, subtraction of a larger value from a smaller value, both smaller than n, will result in a number larger than n. That is, the second digit (in base n) becomes nonzero (in fact, this digit will be equal[2] to $n-1$).

We use this principle to solve the millionaire problem in the following way. Let $\ominus$ denote multiplication with the inverse of the right hand side, which corresponds to subtraction of the right plaintext from the left plaintext. Let $\mathsf{Lift} : \mathbb{Z}^*_{n^2} \to \mathbb{Z}^*_{n^3}$ be the black box lifting procedure that maps $\mathbb{Z}^*_{n^2}$-ciphertexts onto $\mathbb{Z}^*_{n^3}$-ciphertexts while keeping the plaintext intact. Given two ciphertexts c_1 and c_2, we calculate $B = \mathsf{Lift}(c_1 \ominus c_2)$ and $b = \mathsf{Lift}(c_1) \ominus \mathsf{Lift}(c_2)$. Next, we decrypt $A = B \ominus b$. The result can either be different from zero, in which case $c_2 > c_1$, or else 0 and in this case $c_2 \leq c_1$. Aside from this relation, no information is leaked on c_1 and c_2.

This protocol, $\mathcal{P}_{\mathsf{CLMP}}$ is formally presented in Figure 7. It runs with participants A_i holding shares in a t-out-of-k threshold Paillier cryptosystem. The input consists of two ciphertexts c_1 and c_2. While Lift is invoked as a function, in reality it represents an ideal functionality $\mathcal{F}_{\mathsf{Lift}}$.

Protocol $\mathcal{P}_{\mathsf{CLMP}}$

— The authorities calculate $B = \mathsf{Lift}(c_1 \ominus c_2)$, $\mathsf{Lift}(c_1)$ and $\mathsf{Lift}(c_2)$.
— Next, the authorities compute $b = \mathsf{Lift}(c_1) \ominus \mathsf{Lift}(c_2)$ and $A = B \ominus b$.
— Lastly, the authorities decrypt A and output $\mathsf{D}(c_1) \geq \mathsf{D}(c_2)$ if $\mathsf{D}(A) = 0$ and $\mathsf{D}(c_1) < \mathsf{D}(c_2)$ otherwise.

Fig. 7: ciphertext lifting for the millionaire problem.

[2]Unless our lifting procedure is employed and a particular edge case is triggered in which case the digit in question equals $n-2$. See Appendix A for details.

This concludes the description of the ciphertext lifting protocol $\mathcal{P}_{\mathsf{CLMP}}$. What remains to be shown is that $\mathcal{P}_{\mathsf{CLMP}}$ is a secure realization of the ideal protocol $\mathcal{F}_{\mathsf{MP}}$ that solves the millionaire problem. This security holds in the hybrid model as the protocol implicitly relies on a system public key whose matching private key is distributed among the participants ($\mathcal{F}_{\mathsf{SKG}}$), as well on the *accessibility* of this public key, for instance by posting it on the bulletin board ($\mathcal{F}_{\mathsf{BB}}$). Moreover, the zero-knowledge proofs which are required for decryption (and possibly for the lifting procedure, depending on its particular instantiation) are made non-interactive and retain their soundness in the random oracle model. We formalize the random oracle model as an ideal functionality $\mathcal{F}_{\mathsf{RO}}$, along the lines of Hofheinz and Müller-Quade [Hofheinz and Müller-Quade 2004].

THEOREM 3.2. *Protocol $\mathcal{P}_{\mathsf{CLMP}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{MP}}$ in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Lift}})$-hybrid model against non-adaptive adversaries.*

The proof is in Appendix B.2.

3.3. Winner among r Options

Thus far we have proposed a UC-secure protocol for calculating the winner without the vote count in a vote with two options. It turns out that our particular solution extends to multiple options.

The key insight is that a lifted ciphertext $c \in \mathbb{Z}^*_{n^3}$ is homomorphic in the following sense. Let D represent the Damgård-Jurik decryption and let D' be defined as:

$$\mathsf{D}' : \mathbb{Z}^*_{n^3} \to \{0,1\} : c \mapsto \begin{cases} 0 & \text{if } \mathsf{D}(c) = 0 \\ 1 & \text{if } \mathsf{D}(c) \neq 0 \end{cases} .$$

Then we have the following homomorphism for ciphertexts $c_1, c_2 \in \mathbb{Z}^*_{n^3}$ which holds with high probability for random $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_{n^3}$ (where $\vee$ represents the OR operation):

$$\mathsf{D}'(c_1^{r_1} \times c_2^{r_2} \mod n^3) = \mathsf{D}'(c_1) \vee \mathsf{D}'(c_2) .$$

We can exploit this property to solve the *richest* millionaire problem (as opposed to the *richer* millionaire problem, which is necessarily between two participants) by modifying $\mathcal{P}_{\mathsf{CLMP}}$ in the following way. For all pairs $\{c_i, c_j\}$ of vote counts, the authorities invoke $\mathcal{P}_{\mathsf{CLMP}}$ *up to the decryption of A* once on (c_i, c_j) and once on (c_j, c_i). Let the resulting ciphertexts be denoted $A_{i,j}$ and $A_{j,i}$.

Then, for each original ciphertext $c_i \in \mathbf{O}$, the authorities combine all comparison ciphertexts, each of which is randomized by exponentiation with a random $r_j \xleftarrow{\$} \mathbb{Z}_{n^3}$:

$$A_i \leftarrow A_{i,1}^{r_1} \times \ldots \times A_{i,j}^{r_j} \times \ldots \mod n^3$$

That ciphertext c_i whose A_i decrypts to $\mathsf{D}'(A_i) = 0$ encrypts the largest value because this value is as great or greater than that of any c_j. If there are more than one such A_i, then they are tied winners.

The tally-hiding voting system, $\mathcal{P}_{\mathsf{THVS}}$ invokes this modified version of the richest millionaire protocol on all the ciphertexts c_i that encrypt the total number of votes on option O_i. The richest millionaire protocol $\mathcal{P}_{\mathsf{CLMP}}$ will determine the winner of the vote. The security proofs are easily adapted from the two option case.

4. SELF-TALLYING VOTE

A self-tallying voting system is a voting system in which any interested third party can compute the tally after the last vote has been cast. This can be desirable in small-scale votes (*e.g.*: boardroom votes) where all of the voters or a large part of them are simultaneously voting authorities. The price to pay for this increase in efficiency is that one voter can boycott the vote by refusing to participate. The concept was introduced by Kiayias and Yung [Kiayias and Yung 2002] and was later improved on by Groth [Groth 2004b].

We follow the scheme of Kiayias–Yung where the randomizers used for ballot encryption are not entirely random, but designed to cancel out when aggregated. In this way, when all ballots are

put together, the randomizers disappear and we are left with the tally. We overcome the problem in which the last voter can see the tally before he votes by introducing a control voter. This control voter casts a dummy vote only after all voters have cast their votes. It turns out that this control voter is essential for the UC-security of the scheme.

In the systems of both Kiayias–Yung and Groth, all voters are authorities. In our scheme, however, we assume that some voters may not be authorities and that some authorities may not be voters. As long as at least one authority remains honest, no voter's randomizer can be computed.

We describe the protocol for Paillier encryption with the slight modification that the randomizer is generated deterministically from a public seed r to make a cancellation possible. However, it should be noted that our protocol applies to any group in which the discrete logarithm is hard. (Although, eventually, the tally is obtained by computing a discrete logarithm; we selected the Paillier cryptosystem precisely because it offers a subgroup in which this is easy.)

Every voter V_i possesses a secret exponent x_i, such that $\sum_i x_i = 0$. When all ballots are combined, the randomizers cancel:

$$\prod_i (1+n)^{v_i} r^{x_i n} = (1+n)^{\sum_i v_i} \mod n^2 \ .$$

At the start of the protocol, the authorities decide on a base $w \in \mathbb{Z}_n$ for the verification keys. Next, every authority A_j chooses an $x_{i,j}$ for every voter V_i such that $\sum_i x_{i,j} = 0$. This value is sent to voter V_i while $w^{x_{i,j}} \mod n$ is made public. After all authorities have done this, the voter obtains the secret exponent $x_i = \sum_j x_{i,j}$ and any scrutineer can determine that voter's verification key $w_i = \prod_j w^{x_{i,j}} \mod n$.

In order to start voting, the voters must first select a public seed r, for example by taking the hash value of all the messages on the bulletin board up until that point. There is a new public seed for every round of vote casting. It should be noted that $\mathrm{dlog}_w r$ must not be knowable by any subset of authorities, as knowledge of this value can be used to recover individual votes.

A vote is encrypted using the system public key. A non-interactive proof is generated that the vote is correctly formed and the secret exponent was used (these two claims may be proved in a single proof). The result is signed and sent to the bulletin board. A concise overview of this protocol, $\mathcal{P}_{\mathsf{STVS}}$, is presented in Figure 8.

Protocol $\mathcal{P}_{\mathsf{STVS}}$

— Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
— All authorities A_j choose random values $x_{i,j} \xleftarrow{\$} \mathbb{Z}_{n^2}$ for all voters V_i such that $\sum_i x_{i,j} = 0$. They send these values to $\mathcal{F}_{\mathsf{BB}}$ but encrypted with that voter's public key. Also, they publish $w^{x_{i,j}}$ for each $x_{i,j}$.
— The scrutineers check that $\sum_i x_{i,j} = 0$ for all j by calculating $\prod_i w^{x_{i,j}} \stackrel{?}{=} 1$. Also, the scrutineers calculate every voter's verification key $w_i = w^{\sum_j x_{i,j}} = \prod_j w^{x_{i,j}}$.
— All voters V_i read their messages from the authorities and determine their own secret exponent $x_i = \sum_j x_{i,j}$.
— In order to vote, the voters select a public random seed r, for example by querying the random oracle. Next, each voter encrypts his vote using r^{x_i} for randomizer. This encryption is posted to $\mathcal{F}_{\mathsf{BB}}$ along with a proof that the ballot encrypts an allowable vote and that the correct secret exponent was used, and with a signature on the previous two objects. When all real voters have cast their vote, the control voter casts his dummy vote and proves that it does not alter the tally and that it uses the correct secret exponent.
— All participants send a $(\mathbf{read}, sid)$ message to $\mathcal{F}_{\mathsf{BB}}$ and combine all the votes and calculate the tally t.

Fig. 8: self-tallying voting system.

We can only prove that $\mathcal{P}_{\mathsf{STVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ under an additional assumption: the control voter V_n is uncorruptable. The reason is that the honest simulated voters V_i must cast their votes *before* the authorities instruct $\mathcal{F}_{\mathsf{VS}}$ to tally. Therefore, their votes

cannot be made to correspond to the tally unless they cast their votes *after* the invocation of $\mathcal{F}_{\mathsf{VS}}$ — in which case the simulated adversary $\mathcal{A}$ can pick up on the time and order difference. To prevent this, the simulated voters cast random votes, which are then "corrected" by the control voter.

THEOREM 4.1. *$\mathcal{P}_{\mathsf{STVS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ with an additive tallying function against a non-adaptive adversary who has not corrupted the control voter V_n in the $(\mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{RO}})$-hybrid model.*

The proof is in the Appendix B.3.

5. AUTHENTICATED VOTING CREDENTIALS

One of the first proposals for the construction of anonymous voting systems was by using *anonymous credentials* [Chaum 1988]: strings of characters whose membership to a certain class of strings is easy to verify. However, without some trapdoor information pertaining to that class, generating new or other members is intractable.

The idea is as follows. The authorities generate one credential for each eligible voter and securely communicate them to their intended recipients. During voting, the voters log in to the bulletin board via an anonymous channel and cast their votes anonymously. Only those votes that are accompanied by a valid credential are tallied. In an internet setting, the anonymous channel can be approximated by using an anonymization network such as Tor.

However, in recent years credential-based voting systems have fallen out of favor due to their weak security properties. First, the anonymous access to the bulletin board is difficult to realize. The adversary who monitors a target's outgoing traffic as well as the bulletin board's incoming traffic is able to correlate the two and recover the voter's vote. Moreover, to date, there exists no security guarantee for credential-based voting systems. While there do exist UC-secure credential systems [Chase and Lysyanskaya 2006; Camenisch et al. 2010; Trolin 2005], the inclusion of a voting procedure introduces a whole new challenge.

Our contribution is twofold. First, at a conceptual level we introduce the concept of "authenticated voting credentials" for credential-based voting systems. The idea is to protect against the adversary who intercepts the credential or modifies the vote, or who modifies the protocol transcript before presenting it to the verifier. The voter proves his eligibility to vote not by releasing the credential, but by proving knowledge of it. This interactive proof doubles as a MAC on the vote that is cast. The adversary is not able to modify the proof so that it is authenticated by a different vote.

Second, on the level of protocol design, we propose an instantiation of this idea. It is based on Chaumian blind RSA signatures [Chaum 1982], using a credential withdrawal protocol introduced by Ferguson [Ferguson 1993]. A credential consists of a preimage a, a one-way function of the preimage $A = ag^{f(h^a)} \mod n$, and an RSA signature on this value $S = A^{1/v} \mod n$. In these expressions, g and h are publicly known base values in $\mathbb{Z}_n^*$; f is a suitable one-way function; n is an RSA modulus whose factorization is known only to the bank (or, in our case, to the authorities); v is the public RSA exponent and $1/v = v^{-1} \mod \varphi(n)$ is the private RSA exponent.

A simplified version of Ferguson's protocol for credential withdrawal is given by Figure 9. In this figure, H represents a random oracle (instantiated by a hash function) and f a generic one-way function. Compared to Ferguson's original version, the first step has been made non-interactive using the Fiat-Shamir heuristic [Fiat and Shamir 1986].

In a voting system, the private RSA exponent would be distributed among the authorities. The RSA signature is generated using Shoup's protocol [Shoup 2000]. The message from $\mathcal{B}$ to $\mathcal{A}$ actually consists of several messages as each of the authorities communicate their share in $\bar{A}$, the blind signature, to the recipient.

In order to use the credential, the voter must not release the entire string but rather prove knowledge of it. Guillou and Quisquater have proposed an interactive proof of knowledge of an RSA preimage [Guillou and Quisquater 1988], which perfectly matches our purposes. The prover possesses an RSA signature and proves that he knows it while not releasing the signature itself. In

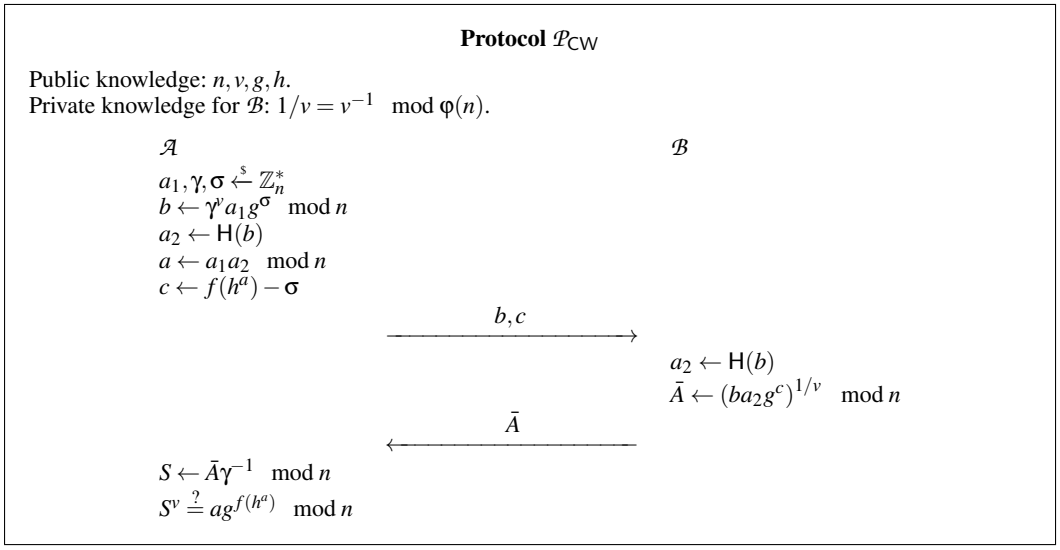


Fig. 9: protocol for credential withdrawal.

Camenisch-Stadler notation:

$$\text{ZKPoK}\{(S) : S^v = A \mod n\} \ .$$

The transcript of the protocol in which the credential is spent, is unlinkable to the transcript of the protocol that created it. The reason is that the transcript of the withdrawal protocol contains no information on the credential. A given withdrawal transcript $(b, c, \bar{A})$ may have created *any* valid credential (a, A, S). To see this, choose $\sigma = f(h^a) - c$; $\gamma = \bar{A}S^{-1} \mod n$ and $a_1 = b\gamma^{-1} g^{-\sigma} \mod n$.

We now describe the entire authenticated voting credential protocol. The protocol is started by invocations of the key generation mechanisms. $\mathcal{F}_{\mathsf{PKG}}$ generates a public and private key pair for encryption and signing for every protocol participant and distributes the public keys such that each participant has a list of public keys of authorities and a list of public keys of eligible voters. $\mathcal{F}_{\mathsf{SKG}}$ generates an RSA public key and distributes shares in the private key to the authorities along Shoup's protocol [Shoup 2000].

In the second phase of the protocol, the voters withdraw their credential. They initiate the credential withdrawal protocol $\mathcal{P}_{\mathsf{CW}}$ (in the role of $\mathcal{A}$) and sign their first message. The authorities (in the role of $\mathcal{B}$) assign credentials only to eligible voters — whom they can tell from non-eligible voters by the signature which must match a public key in the list of eligible voter public keys. Each participant performs the instructions of $\mathcal{B}$ from $\mathcal{P}_{\mathsf{CW}}$ separately, returning not the signature but the signature share on A and encrypting it with that voter's public key, along with a proof of correct exponentiation. If at least t authorities do this honestly, the voter will be able to reconstruct the signature $S = A^{1/v} \mod n$ which is an essential part of the credential.

At this point, every eligible voter (or those who have initiated the withdrawal process) is in possession of a valid credential $C = (a, A, S)$ which is unlinkable to the transcript of the withdrawal process that created it. During the voting phase, the voters log in to the bulletin board anonymously and post one message containing all of the following. First, their vote v_i. Second, the credential preimage a. And last, a non-interactive proof of knowledge of $S = A^{1/v} = (a g^{f(h^a)})^{1/v} \mod n$. This non-interactive proof is generated by applying the Fiat-Shamir heuristic to the proof of RSA signature knowledge $\mathcal{P}_{\mathsf{RSA}}$, where the challenge value e is computed as the hash of the proof claim, the proof's first message, and the vote that is cast:

$$e = \mathsf{H}(A \parallel D \parallel v_i) \ .$$

Anyone can compute the tally by counting only those votes that are accompanied by a valid and unique credential. A credential cannot be reused as the value a is fixed. Only eligible voters possess credentials because the authorities only assume their role in the credential withdrawal protocol when eligible voters request a credential (and they make sure not to issue multiple credentials to any one voter). Since access to the bulletin board is anonymous and since the credentials cannot be linked to the withdrawal process that created them, voter anonymity is preserved. Lastly, the votes that were cast conform to the credential holder's intention because otherwise the proof will not check out. The protocol is formally presented in Figure 10.

Protocol $\mathcal{P}_{\mathsf{AVCS}}$

— Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
— All voters V_i that wish to vote post to $\mathcal{F}_{\mathsf{BB}}$ the pair of values b and c, which constitutes the first message in $\mathcal{P}_{\mathsf{CW}}$. This pair is signed by the voter.
— All authorities A_j, upon seeing this initial message from V_i, compute $\bar{A}_j = (b\mathsf{H}(b)g^c)^{s_j} \mod n$ (where s_j is the share in the private exponent for authority A_j). They send this value, along with a proof of correct exponentiation to $\mathcal{F}_{\mathsf{BB}}$ after encrypting it with V_i's public key.
— All voters V_i compute the signature in their credentials by calculating $\bar{A} \leftarrow \prod_j \bar{A}_j^{2\Delta s_j \mu_j} \mod n$ where μ_j is the Lagrange coefficient, and then $S \leftarrow \bar{A}\gamma^{-1} \mod n$.
— All voters cast their votes anonymously by sending (v_i, a, Z) to $\mathcal{F}_{\mathsf{BB}}$, where Z is the transcript of the proof of knowledge of $S = A^{1/v} = (ag^{f(h^a)})^{1/v} \mod n$ made non-interactive by setting the challenge value to $e = \mathsf{H}(A \parallel D \parallel v_i)$.
— Anyone, including the scrutineers, can compute the tally by sending a $(\textbf{read}, sid)$ message to $\mathcal{F}_{\mathsf{BB}}$ and tallying the votes from all the messages (v, a, Z) where Z is a valid proof for v and a (and where a was not used before).

Fig. 10: authenticated voting credential system.

We note that this protocol is not *fair*: voters can observe the current tally before casting their own vote. Knowledge of this partial tally may influence the content of their vote or even whether or not they choose to cast it. This drawback is addressed by universal composability. If the times at which the votes are cast, can influence which votes are cast, then the protocol adversary $\mathcal{A}$ can exploit this property and generate votes where the tallies are significantly different from the tallies that would have been produced by $\mathcal{F}_{\mathsf{VS}}$.

In order to make the voting system fair — and, hence, in order for the system to possibly be universally composable — voters must not be able to read the bulletin board during the voting phase. Conceptually, we modify $\mathcal{F}_{\mathsf{BB}}$ to offer this property. In particular: if $\mathcal{F}_{\mathsf{BB}}$ receives a $(\textbf{close}, sid)$ from at least one honest authority, $\mathcal{F}_{\mathsf{BB}}$ starts ignoring all $(\textbf{read}, sid)$ requests. When at least one honest authority sends a $(\textbf{open}, sid)$ message, $\mathcal{F}_{\mathsf{BB}}$ starts responding again. This can be implemented on top of $\mathcal{F}_{\mathsf{BB}}$ by introducing an interfacing layer that relays messages between $\mathcal{F}_{\mathsf{BB}}$ and the external environment but blocks **read** messages when it is not readable.

The adversary who can read, block or falsify all incoming traffic to $\mathcal{F}_{\mathsf{BB}}$ breaks the security of the protocol because he can trivially see which vote is coming from which voter. We can only prove security under a far weaker adversarial model. In this model, the adversary can no longer read, block or falsify messages sent by voters to the bulletin board. In other words, except for instructing the corrupted parties, the adversary has become *passive*. In order to account for this adversary, we modify the specification of the bulletin board functionality $\mathcal{F}_{\mathsf{BB}}$ to instantly store any message by any participant without reference to the sender and without letting it pass through the adversary. In this modified $\mathcal{F}_{\mathsf{BB}}$ model, we are able to prove the UC-security and universal verifiability of $\mathcal{P}_{\mathsf{AVCS}}$.

THEOREM 5.1. *Protocol $\mathcal{P}_{\mathsf{AVCS}}$ is a secure and universally verifiable realization of $\mathcal{F}_{\mathsf{VS}}$ with an additive tallying function in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{F}_{\mathsf{PKG}})$-hybrid model for a passive, non-adaptive*

adversary corrupting up to $k-t$ authorities, where k is the number of authorities and t is the threshold of honest authorities.

The proof is in Appendix B.4.

Universal composability does not require authenticated voting credentials. As long as the bulletin board offers anonymous, append-only access and is unreadable during voting but otherwise readable by all participants, any credential system can be used for UC-secure voting. On the other hand, if the adversary is able to control what the bulletin board shows to readers during the tallying process, then authenticated voting credentials are necessary for UC-security. More importantly, they are necessary for universal verifiability. Since the adversary controls the transcript that is verified by the verifier, he could otherwise pretend the (unauthenticated) credentials were used to cast different votes and so convince the verifier of a false tally. This is impossible if the credentials are authenticated by the votes.

6. CONCLUSION

In this paper we have introduced three new techniques for electronic voting as well as a unifying framework for their analysis. In particular, we have proposed a tally-hiding voting system, a self-tallying voting system and an authenticated voting credential system and we have proved the UC-security of each of these systems as well as their universal verifiability.

The UC-security of the systems is proved under various adversarial models. Our tally-hiding system is secure against the strongest adversary: one who can observe, block or falsify any voter's messages and corrupt any number of voters and up to $k-t$ authorities, where k is the number of authorities and t is the threshold of honest authorities. In the self-tallying vote, the adversary is nearly as strong. The only additional constraint is that there is a control voter who casts the last dummy vote and who cannot be corrupted. In both cases, he is allowed to read, block and falsify voters' messages, extending the power of the adversary compared to Groth's model [Groth 2004b]. On the other hand, Groth's model includes protection against adaptive adversaries in the erasure and in the erasure-free model. The adversary in the authenticated voting credential system is far weaker as he is not able to observe voters' messages — not even indirectly by observing the bulletin board as this may be closed for reading. In all cases, we assume the adversary is computationally bounded and that his power to corrupt is somehow constrained.

We offer a formalism of universal verifiability which is compatible with universal composability in two senses: first, the verifier (from the universal verifiability paradigm) interacts with the adversary (from the universal composability paradigm); second, universally verifiable protocols are composable just as universally composable protocols are. While universal verifiability is independent of the ideal bulletin board functionality, it is not independent of the participant key generation functionality nor of the random oracle functionality. The verifier $\mathcal{V}$ needs to verify the authenticity of, at the very least, the voters' messages. The random oracle is necessary to make the proofs non-interactive. While the adversary-simulator $\mathcal{S}$ may simulate a random oracle to convince a real protocol adversary $\mathcal{A}$ of false claims, he cannot convince a skeptical verifier in the same way. In order to convince a verifier of the correctness of a protocol, the adversary must invoke a random oracle that is trusted by the verifier. To make this non-interactive, a cryptographic hash function is a reasonable approximation.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their useful comments and countless colleagues for fruitful discussions and critical proof-readings. This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). Alan Szepieniec is supported by a doctoral grant from the Agency for Innovation by Science and Technology in Flanders (IWT). In addition, this work was supported by the Flemish Government, FWO Beveiligde Uitvoering G.0130.13 and by the European Commission through the ICT programme under contract H2020-ICT-2014-645421 ECRYPT CSA, H2020-MSCA-ITN-2014-643161 ECRYPT NET and FP7-ICT-2013-645622 PRACTICE.

REFERENCES

Ben Adida. 2006. *Advances in cryptographic voting systems*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Carsten Baum, Ivan Damgård, and Claudio Orlandi. 2014. Publicly Auditable Secure Multi-Party Computation. *IACR Cryptology ePrint Archive* 2014 (2014), 75.

Josh Benaloh. 1987. *Verifiable secret-ballot elections.* Ph.D. Dissertation. Yale University.

Josh Cohen Benaloh and Dwight Tuinstra. 1994. Receipt-free secret-ballot elections (extended abstract). In *STOC*, Frank Thomson Leighton and Michael T. Goodrich (Eds.). ACM, 544–553.

Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. 2010. Credential Authenticated Identification and Key Exchange. In *CRYPTO (Lecture Notes in Computer Science)*, Tal Rabin (Ed.), Vol. 6223. Springer, 255–276.

Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*. IEEE Computer Society, 136–145.

Ran Canetti and Rosario Gennaro. 1996. Incoercible Multiparty Computation (extended abstract). In *FOCS*. IEEE Computer Society, 504–513.

Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO (Lecture Notes in Computer Science)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, 78–96.

David Chaum. 1982. Blind Signatures for Untraceable Payments. In *CRYPTO*, David Chaum, Ronald L. Rivest, and Alan T. Sherman (Eds.). Plenum Press, New York, 199–203.

David Chaum. 1988. Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA, See Günther [1988], 177–182.

Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. 2010. On Some Incompatible Properties of Voting Schemes. In *Towards Trustworthy Elections*. 191–199.

Josh D Cohen. 1986. *Improving privacy in cryptographic elections*. Citeseer.

Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO (Lecture Notes in Computer Science)*, Yvo Desmedt (Ed.), Vol. 839. Springer, 174–187.

Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. 1996. Multi-Autority Secret-Ballot Elections with Linear Work. In *EUROCRYPT (Lecture Notes in Computer Science)*, Ueli M. Maurer (Ed.), Vol. 1070. Springer, 72–83.

Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT (Lecture Notes in Computer Science)*, Walter Fumy (Ed.), Vol. 1233. Springer, 103–118.

Ivan Damgård and Mads Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography (Lecture Notes in Computer Science)*, Kwangjo Kim (Ed.), Vol. 1992. Springer, 119–136.

Ivan Damgård and Mads Jurik. 2002. Client/Server Tradeoffs for Online Elections, See Naccache and Paillier [2002], 125–140.

Niels Ferguson. 1993. Single Term Off-Line Coins. In *EUROCRYPT (Lecture Notes in Computer Science)*, Tor Helleseth (Ed.), Vol. 765. Springer, 318–328.

Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO (Lecture Notes in Computer Science)*, Andrew M. Odlyzko (Ed.), Vol. 263. Springer, 186–194.

Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. 1992. A Practical Secret Voting Scheme for Large Scale Elections. In *AUSCRYPT (Lecture Notes in Computer Science)*, Jennifer Seberry and Yuliang Zheng (Eds.), Vol. 718. Springer, 244–251.

Jens Groth. 2004a. Efficient Maximal Privacy in Boardroom Voting and Anonymous Broadcast. In *Financial Cryptography (Lecture Notes in Computer Science)*, Ari Juels (Ed.), Vol. 3110. Springer, 90–104.

Jens Groth. 2004b. Evaluating Security of Voting Schemes in the Universal Composability Framework. In *ACNS (Lecture Notes in Computer Science)*, Markus Jakobsson, Moti Yung, and Jianying Zhou (Eds.), Vol. 3089. Springer, 46–60.

Louis C. Guillou and Jean-Jacques Quisquater. 1988. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory, See Günther [1988], 123–128.

Christoph G. Günther (Ed.). 1988. *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*. Lecture Notes in Computer Science, Vol. 330. Springer.

Dennis Hofheinz and Jörn Müller-Quade. 2004. Universally Composable Commitments Using Random Oracles. In *TCC (Lecture Notes in Computer Science)*, Moni Naor (Ed.), Vol. 2951. Springer, 58–76.

Hugo L. Jonker and Erik P. de Vink. 2006. Formalising Receipt-Freeness. In *ISC (Lecture Notes in Computer Science)*, Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel (Eds.), Vol. 4176. Springer, 476–488.

Aggelos Kiayias and Moti Yung. 2002. Self-tallying Elections and Perfect Ballot Secrecy, See Naccache and Paillier [2002], 141–158.

David Naccache and Pascal Paillier (Eds.). 2002. *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*. Lecture Notes in Computer Science, Vol. 2274. Springer.

Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. 1999. An Improvement on a Practical Secret Voting Scheme. In *ISW (Lecture Notes in Computer Science)*, Masahiro Mambo and Yuliang Zheng (Eds.), Vol. 1729. Springer, 225–234.

Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT (Lecture Notes in Computer Science)*, Jacques Stern (Ed.), Vol. 1592. Springer, 223–238.

Kazue Sako and Joe Kilian. 1995. Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth. In *EUROCRYPT (Lecture Notes in Computer Science)*, Louis C. Guillou and Jean-Jacques Quisquater (Eds.), Vol. 921. Springer, 393–403.

Victor Shoup. 2000. Practical Threshold Signatures. In *EUROCRYPT (Lecture Notes in Computer Science)*, Bart Preneel (Ed.), Vol. 1807. Springer, 207–220.

Mårten Trolin. 2005. A Universally Composable Scheme for Electronic Cash. In *INDOCRYPT (Lecture Notes in Computer Science)*, Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan (Eds.), Vol. 3797. Springer, 347–360.

Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *FOCS*. IEEE Computer Society, 160–164.

Received March 2015; revised June 2015; accepted June 2015

A. CIPHERTEXT LIFTING (AND THE MILLIONAIRE PROBLEM)

We now describe an alternative to Damgård and Jurik's ciphertext lifting procedure [Damgård and Jurik 2002]. One key ingredient is that the authorities decide on an access structure Λ (a set of t indices indentifying the authorities that will perform the computation) in advance. When such an access structure has been decided, each of the authorities uses their private exponent s_i to compute a share c_i starting from a ciphertext c as follows:

$$c_i \leftarrow c^{4\Delta^2 s_i \prod_{j\in\Lambda\setminus\{i\}} \frac{-j}{i-j}} \mod n^2 \ .$$

These shares constitute a multiplicative sharing modulo n^2 as the following equation is guaranteed to hold thanks to Lagrange:

$$(1+n)^m = \prod_{i\in\Lambda} c_i \mod n^2 \ .$$

In the next step of the protocol, each authority divides their share into two digits base n: $c_i = a_i + b_i n$ where $a_i, b_i < n$. The a_i are published; the b_i are kept secret. It is easy to see that the a_i do not leak information on the plaintext m. Moreover, the proof of correct exponentiation which is used to prove the correctness of a decryption share, can be performed modulo n rather than modulo n^2 to yield a zero-knowledge proof that a_i is correctly formed.

The authorities proceed to compute an additive sharing modulo n of the plaintext as follows. Each authority in the access structure computes their share u_i:

$$u_i \leftarrow b_i \prod_{j\in\Lambda\setminus\{i\}} a_j \mod n \ .$$

Anyone can compute $1+nr = \prod_{i\in\Lambda} a_i \mod n^2$. At this point, the following equality is guaranteed to hold:

$$m = r + \sum_{i\in\Lambda} u_i \mod n \ .$$

In addition to computing their shares u_i, the authorities commit to their shares by publishing $\mathsf{E}(u_i)$. Moreover, they choose some randomness to encrypt r as well. At this point the authorities

use the regular threshold decryption procedure to prove that the value

$$c^{-1}\mathsf{E}(r)\prod_{i\in\Lambda}\mathsf{E}(u_i) \mod n^2$$

does indeed decrypt to 0 — implying that the commitments to their shares are correct.

The next step is to obtain an additive sharing of the message *without modular reduction*. This step only works by leaking the most significant few bits of the plaintext. While this property is not desirable (as it precludes UC-security), we argue that this is a minor compromise considering the size of the modulus and the fact that this lifting procedure's most pertinent use case is comparing plaintexts — *i.e.* discovering certain information about the most significant bits.

We use a safety parameter $\varsigma \geq 2$ to divide the range $\mathbb{Z}_n$ into a *secret zone* $S = [0; \frac{n}{\varsigma}) \cap \mathbb{Z}_n$ and a *danger zone* $D = [n - \frac{n}{\varsigma}; n) \cap \mathbb{Z}_n$. These sets have the property that any secret element added to a non-dangerous element is guaranteed not to overflow. Symbolically:

$$\forall s \in S, d \in \mathbb{Z}_n \backslash D \,.\, s + d < n \ .$$

We use this principle to find another sharing of the plaintext whose equation holds with and without modular reduction. In particular, the sum of all the secret shares belongs to the safe zone, and the new public remainder r' does not belong to the danger zone. In particular, each authority in the access structure chooses a new share $u'_i \xleftarrow{\$} \left\{0, \ldots, \left\lfloor \frac{n}{\varsigma t} \right\rfloor\right\}$ at random. (Bear in mind that $t = |\Lambda|$.) The difference $\delta_i = u_i - u'_i$ is made public along with commitments such that anyone can verify that $\mathsf{E}(u_i) = \mathsf{E}(\delta_i)\mathsf{E}(u'_i) \mod n^2$. The new remainder r' is computed as $r' = r + \sum \delta_i \mod n$. Thus we have an additive sharing which is guaranteed with high probability not to overflow:

$$r' + \sum_{i\in\Lambda} u'_i < n \ .$$

Of course, there is a chance that the procedure as described above might overflow after all and produce $\mathsf{E}(m+n)$ rather than $\mathsf{E}(m)$. This occurs whenever $m < \sum_{i\in\Lambda} u'_i$; the probability of this event is $\frac{1}{2\varsigma}$. It is possible to make this probability small by increasing the safety parameter ς, but this comes at the cost of leaking more information on the plaintext.

Whenever this event occurs, it will be signalled by $r' \in D$ (although $r' \in D$ does not automatically imply $m < \sum_{i\in\Lambda} u'_i$). If the authorities discover that $r' \in D$ and if it is known that $m \notin D$, then they can circumvent the failure by adding $\lfloor \frac{n}{\varsigma} \rfloor$ to r', performing the reduction modulo n, lifting, and homomorphically subtracting $\lfloor \frac{n}{\varsigma} \rfloor$ from the lifted ciphertext.

Of course, this does leave open the question what to do when it is not known whether $m \notin D$ and r' is found to lie in D. We would argue that despite the lack of robustness against errors with respect to all plaintexts, this lifting procedure does offer a practical solution to most use cases, especially considering that ς can be balanced against the known distribution of m. More importantly, however, this drawback does not seem to impact the solution to the millionaire problem. We now demonstrate this.

Without loss of generality, we can assume that $m_1 \notin S$ and $m_2 \notin S$, because we can homomorphically add $\lfloor \frac{n}{\varsigma} \rfloor$ to both plaintexts at no cost. However, if m_1 or m_2 happens to lie in D, then the addition of $\lfloor \frac{n}{\varsigma} \rfloor$ will cause overflow, breaking the protocol. If this is the case then the solution is to use a larger modulus n.

The protocol performs lifting on three numbers: $C_1 = \mathsf{Lift}(c_1), C_2 = \mathsf{Lift}(c_2)$ and $B = \mathsf{Lift}(c_1 \ominus c_2)$. The first two are guaranteed to succeed with this lifting procedure. However, the ciphertext B may encode a plaintext that is larger than n, depending on the particular relation between m_1 and m_2. In particular, with the application of the addition trick from the paragraph before last, we know that B

satisfies:

$$\mathsf{D}(B) = \begin{cases} m_1 - m_2 & \text{if} \quad m_1 \geq m_2 \\ 2n + m_1 - m_2 & \text{if} \quad m_2 - m_1 \in S \backslash \{0\} \\ n + m_1 - m_2 & \text{else} \end{cases} .$$

However, $b = C_1 \ominus C_2$ will satisfy:

$$\mathsf{D}(b) = \begin{cases} m_1 - m_2 & \text{if} \quad m_1 \geq m_2 \\ n^2 + m_1 - m_2 & \text{if} \quad m_2 - m_1 \in S \backslash \{0\} \\ n^2 + m_1 - m_2 & \text{else} \end{cases} .$$

And hence the difference $A = B \ominus b$ will satisfy:

$$\mathsf{D}(A) = \begin{cases} 0 & \text{if} \quad m_1 \geq m_2 \\ 2n & \text{if} \quad m_2 - m_1 \in S \backslash \{0\} \\ n & \text{else} \end{cases} .$$

Which implies in particular that the inequality test at the conclusion of the millionaire problem solution still holds: $\mathsf{D}(A) \neq 0 \Rightarrow m_1 < m_2$ and conversely $\mathsf{D}(A) = 0 \Rightarrow m_1 \geq m_2$. In order to further reduce the leakage of information, the ciphertext A must be raised to an unknown and random (but provably nonzero) exponent prior to decryption.

B. UNIVERSALLY COMPOSABLE AND VERIFIABLE VOTING SYSTEMS

B.1. Security and Verifiability of the Tally-Hiding Voting System

PROOF. of Theorem 3.1. *Correctness.* Correctness follows from construction.

UC-security. We show that no environment $\mathcal{E}$ can tell the difference between an execution of the protocol under attack by a real protocol adversary $\mathcal{A}$ or an invocation of the ideal functionality $\mathcal{F}_{\mathsf{VS}}$ under attack by an adversary-simulator $\mathcal{S}$.

The adversary-simulator $\mathcal{S}$ runs with dummy voters $\hat{V}_i$ and dummy authorities $\hat{A}_i$. The honest dummy voters relay their inputs from $\mathcal{E}$ directly to the ideal functionality $\mathcal{F}_{\mathsf{VS}}$. The corrupted dummy voters and authorities are controlled by the adversary-simulator $\mathcal{S}$. At the end of the experiment, $\mathcal{E}$ reads the outputs of the voters.

The adversary-simulator simulates an execution of the protocol $\mathcal{P}_{\mathsf{THVS}}$ and simulates the real protocol adversary $\mathcal{A}$. The protocol runs with simulated voters V_i and authorities A_j, some of which are controlled by the real process adversary $\mathcal{A}$ and the others of which are controlled by the adversary-simulator $\mathcal{S}$. The set of corrupt dummy participants (i.e.: participants $\hat{V}_i$ or $\hat{A}_i$ that are controlled by $\mathcal{S}$) matches the set of corrupt simulated participants (i.e.: controlled by $\mathcal{A}$). The adversary-simulator $\mathcal{S}$ proceeds as follows.

— $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{E}$.
— $\mathcal{S}$ starts by simulating the invocation of $\mathcal{F}_{\mathsf{PKG}}$ and $\mathcal{F}_{\mathsf{SKG}}$. $\mathcal{S}$ uses $\mathcal{F}_{\mathsf{BB}}$ throughout. As a result, $\mathcal{S}$ knows *all* private keys and shares in the system private key, in addition to all public keys.
— If one of the simulated voters V_i that is controlled by the real process adversary $\mathcal{A}$ casts a ballot b_i and a proof p_i, the adversary-simulator $\mathcal{S}$ checks the proof and if it is valid, decrypts b_i to obtain V_i's vote, v_i. Next, $\mathcal{S}$ has $\hat{V}_i$ send $(\textbf{vote}, sid, v_i, \hat{V}_i)$ to $\mathcal{F}_{\mathbf{VS}}$. $\mathcal{S}$ follows up with a $(\textbf{no-block}, sid, \hat{V}_i)$ message.
— If the adversary-simulator $\mathcal{S}$ receives $(\textbf{vote}, sid, \hat{V}_i)$ from $\mathcal{F}_{\mathsf{VS}}$ pertaining to an uncorrupted voter $\hat{V}_i$, $\mathcal{S}$ simulates V_i's vote in the simulated protocol by casting a zero vote $v_i = (0,0)$ to the bulletin board.
— If the authorities in the simulated protocol invoke $\mathcal{F}_{\mathsf{MP}}$, $\mathcal{S}$ lets all $\hat{A}_i$'s under his control send $(\textbf{tally}, sid)$ to $\mathcal{F}_{\mathsf{VS}}$, which will produce a tally t. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{MP}}$ for the simulated protocol by presenting the authorities with an answer conformant with t. The dummy voters $\hat{V}_i$ receive the tally from $\mathcal{F}_{\mathsf{VS}}$.

The real protocol adversary $\mathcal{A}$ has no way to determine whether or not he is being simulated by $\mathcal{S}$. The only messages that might indicate that he is being simulated, are the encryptions of votes that are not conformant to the winner coming from $\mathcal{F}_{\mathsf{MP}}$. However, if $\mathcal{A}$ can tell that the vote encryptions do not conform to the tally, then he either breaks the semantic security of the cryptosystem or has access to more than $k-t$ shares in the private system key.

Similarly, the environment $\mathcal{E}$ has no way of detecting whether there is a adversary-simulator $\mathcal{S}$ between himself and the real protocol adversary $\mathcal{A}$. The tallies on the tapes of the voters V_i and authorities A_j are the same in both cases. All $\mathcal{E}$'s queries are answered honestly by a real protocol adversary $\mathcal{A}$ who has no way of knowing whether or not he is being simulated. Moreover, $\mathcal{E}$ is subject to the same computational constraints, which in particular imply that $\mathcal{E}$ cannot break the semantic security of the cryptosystem either.

Universal verifiability. We show this by constructing a verifier $\mathcal{V}$ which verifies an execution of the protocol.

$\mathcal{V}$ does not need to verify the correct execution of $\mathcal{F}_{\mathsf{PKG}}$ or $\mathcal{F}_{\mathsf{SKG}}$ as these are trusted functionalities. The ballots that are cast are accompanied by zero-knowledge proofs which prove their correct composition. Moreover, they are signed, which authenticates the eligibility of their originator. $\mathcal{V}$ can calculate the aggregates of the vote encryptions himself and verify that the input to $\mathcal{F}_{\mathsf{MP}}$ is the correct input. He does not need to verify the execution of $\mathcal{F}_{\mathsf{MP}}$ as it is a trusted functionality. Lastly, $\mathcal{V}$ checks whether the tally sent to the bulletin board conforms with the answer from $\mathcal{F}_{\mathsf{MP}}$. □

B.2. Security and Verifiability of the Millionaire Problem Solution based on Lifting

PROOF. of Theorem 3.2. *Correctness.* Correctness follows from construction. Let $d = m_1 - m_2$, then we have:

$$A = B \ominus b = \mathsf{Lift}(c_1 \ominus c_2) \ominus (\mathsf{Lift}(c_1) \ominus \mathsf{Lift}(c_2))$$

$$= \mathsf{Lift}(\mathsf{E}(m_1) \ominus \mathsf{E}(m_2)) \ominus (\mathsf{Lift}(\mathsf{E}(m_1)) \ominus \mathsf{Lift}(\mathsf{E}(m_2)))$$

$$= \mathsf{E}(m_1 - m_2 \mod n) \ominus \mathsf{E}(m_1 - m_2 \mod n^2)$$

$$\mathsf{E}(n + d \mod n) \ominus \mathsf{E}(n^2 + d \mod n^2) = \mathsf{E}(rn) \ .$$

If $m_1 < m_2$, then $r \neq 0$ (and in fact $r = 1$). On the other hand, if $m_1 \geq m_2$, then $r = 0$.

UC-security. We demonstrate that for every adversary $\mathcal{A}$ that attacks the protocol, there exists an adversary-simulator $\mathcal{S}$ that attacks the ideal functionality with equal success such that any computationally bounded external environment $\mathcal{E}$ cannot determine whether it is interacting with $\mathcal{A}$ and the protocol, or with $\mathcal{S}$ and the ideal functionality.

In the beginning of the experiment, $\mathcal{E}$ reads the system public key. He then chooses a pair of plaintexts m_1 and m_2 and encrypts them using the public key to obtain c_1 and c_2. These values are sent to all participants. At the end of the experiment, the participants communicate the result to $\mathcal{E}$. This implicit reliance on a pre-distributed private key is made explicit if the protocol invokes $\mathcal{F}_{\mathsf{SKG}}$. The implicit reliance that this public key is accessible by $\mathcal{E}$ is made explicit if the protocol uses $\mathcal{F}_{\mathsf{BB}}$. Since the protocol is only a subprotocol within $\mathcal{P}_{\mathsf{THVS}}$ which does use these ideal functionalities, this is not a problem. However, UC-security only holds in the hybrid model.

The adversary-simulator $\mathcal{S}$ controls the corrupt participants. Moreover, $\mathcal{S}$ runs a simulation of the protocol $\mathcal{P}_{\mathsf{CLMP}}$ with participants A_i, mapping corrupt $\hat{A}_i$ to corrupt A_i and simulating honest A_i as well as the adversary $\mathcal{A}$ that attacks the protocol. Lastly, $\mathcal{S}$ controls the random oracle, which means that the simulated parties A_i can create valid proofs for false claims.

$\mathcal{S}$ behaves as follows:

— $\mathcal{S}$ forwards all messages between $\mathcal{E}$ and $\mathcal{A}$.
— The simulated participants A_i hold the same ciphertexts c_1 and c_2.
— If $\mathcal{S}$ controls t or more participants $\hat{A}_i$, then $\mathcal{S}$ is able to decrypt the ciphertexts and obtain plaintexts m_1, m_2. Otherwise, he chooses plaintexts m_1 and m_2 at random.

— When the participants A_i run the Lift subprocedure, they raise ℓ_{i-1} to a random secret power and generate a non-interactive proof that this exponentiation conforms to their share verification key (v, v_i) using the corrupt random oracle.
— When the participants A_i decrypt A, they produce random decryption shares consistent with 0 (i.e.: produces 0 upon combination) if $m_1 \geq m_2$ and rn otherwise, where r is a random value from $\mathbb{Z}_n$. Since $\mathcal{S}$ controls the random oracle, the proofs of "correct" decryption are easy to falsify.

The real protocol adversary $\mathcal{A}$ has no way of discovering that he is being simulated by $\mathcal{S}$. The only messages that might indicate that he is being simulated, are the ciphertexts c_1, c_2 and intermediate encryptions having to do with lifting that do not conform to the result coming back from $\mathcal{F}_{\mathsf{MP}}$. However, if $\mathcal{A}$ can infer this from the ciphertexts, then he breaks the cryptosystem's semantic security. Similarly, if he can glean from the transcripts of the zero-knowledge proofs that the intermediate values do not conform to the result returned from $\mathcal{F}_{\mathsf{MP}}$, then he breaks the computational zero-knowledge property. Since $\mathcal{A}$ trusts the random oracle, he has no reason to assume the zero-knowledge proofs are incorrect.

Similarly, the environment $\mathcal{E}$ has no way of discovering that there is an adversary-simulator $\mathcal{S}$ between himself and the protocol-adversary $\mathcal{A}$, who also has no idea that such an $\mathcal{S}$ exists. Moreover, since $\mathcal{E}$ does not have access to the participants' decryption shares, he cannot discover that the ciphertexts and intermediate ciphertexts do not conform to the result of the protocol without breaking the cryptosystem's semantic security. Moreover, this result is guaranteed to match the parties' inputs as $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{MP}}$.

Universal verifiability. Every step in the protocol has an associated message on the bulletin board. Except for lifting and decryption, the verifier can check the new values by calculating them himself. Lifting is a trusted ideal functionality. As for decryption, every step in this subprotocol is accompanied by efficiently verifiable zero-knowledge proofs which ensures the validity of that one step. □

B.3. Security and Verifiability of the Self-Tallying Vote

PROOF. of Theorem 4.1. *Correctness.* Correctness follows from construction. Let $\mathsf{E}(m,r)$ denote the encryption using the system public key of the message m using randomizer r. Let $\oplus$ denote the operation on ciphertexts that homomorphically corresponds to addition. The aggregation of the encrypted votes is given by:

$$\mathsf{E}(v_1,r_1)\oplus\ldots\oplus\mathsf{E}(v_n,r_n)=\mathsf{E}\left(\sum_{i=1}^{n}v_i,\prod_{i=1}^{n}r^{x_i}\right)=\mathsf{E}\left(t,r^{\Sigma_{i=1}^{n}x_i}\right)=\mathsf{E}\left(t,r^0\right)=\mathsf{E}(t,1)$$

UC-security. We demonstrate that for every adversary $\mathcal{A}$ who attacks the protocol, there exists an adversary-simulator $\mathcal{S}$ who attacks the ideal functionality with equal success, such that no computationally bounded external environment $\mathcal{E}$ can tell whether he is interacting with an execution of the protocol and a real adversary $\mathcal{A}$ attacking it, or with an invocation of the ideal functionality and an adversary-simulator $\mathcal{S}$ that attacks it.

The adversary-simulator runs with dummy voters $\hat{V}_i$ and dummy authorities $\hat{A}_j$. The dummy voters $\hat{V}_1,\ldots,\hat{V}_{n-1}$ receive their input from $\mathcal{E}$. The corrupt voters and authorities are controlled by $\mathcal{S}$ and the honest voters and authorities relay their inputs to $\mathcal{F}_{\mathsf{STVS}}$. At the end of the experiment, $\mathcal{E}$ reads the outputs of the dummy voters. Throughout the experiment, $\mathcal{E}$ can query the adversary. After the experiment, $\mathcal{E}$ is allowed a polynomial-time computation before he must decide whether he was interacting with $\mathcal{A}$ or $\mathcal{S}$.

The adversary-simulator $\mathcal{S}$ simulates an execution of the protocol between simulated authorities A_j and simulated voters V_i, along with the (simulated) real protocol adversary $\mathcal{A}$. The set of corrupt simulated voters and authorities matches the set of corrupt dummy voters and authorities. The adversary-simulator $\mathcal{S}$ proceeds as follows:

— $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{E}$.

— $\mathcal{S}$ starts by simulating the invocation of $\mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{SKG}}$ and simulates $\mathcal{F}_{\mathsf{BB}}$ throughout. As a result, $\mathcal{S}$ knows all participants' private keys as well as the system private key.
— $\mathcal{S}$ allows the authorities A_j to generate secret exponents x_i for all voters V_i. $\mathcal{S}$ is able to decrypt these messages and obtain the x_i.
— If one of the corrupt voters V_i casts their vote, $\mathcal{S}$ checks the proofs and decrypts the vote v_i and instructs V_i to send $(\textbf{vote}, sid, v_i, \hat{V}_i)$ to $\mathcal{F}_{\mathsf{VS}}$. $\mathcal{S}$ follows up with a $(\textbf{no-block}, sid, \hat{V}_i)$ message.
— On receiving $(\textbf{vote}, sid, \hat{V}_i)$ from $\mathcal{F}_{\mathsf{VS}}$ pertaining to an honest voter $\hat{V}_i$, $\mathcal{S}$ instructs V_i to cast a random vote. In order to generate sound proofs, $\mathcal{S}$ simulates the random oracle $\mathcal{F}_{\mathsf{RO}}$ to generate suitable outputs.
— When all the corrupt voters V_i have cast their votes, $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{VS}}$ and receives the tally t. $\mathcal{S}$ determines the current tally t' in the simulated protocol and instructs the control voter V_n to cast a vote of the form $\mathsf{E}(t-t')$. Again, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{RO}}$ so as to generate sound proofs.

The only ways in which $\mathcal{A}$ can discover that he is being simulated by $\mathcal{S}$ is by discovering that the encrypted votes (from the honest voters as well as from the control voter) are not valid votes. In order to come to this conclusion, $\mathcal{A}$ would have to break the semantic security of the cryptosystem or else discover that the challenge values in the zero-knowledge proofs are not random. But as $\mathcal{A}$ has bounded computational power and trusts $\mathcal{F}_{\mathsf{RO}}$, this cannot happen.

The environment $\mathcal{E}$ gets to communicate with a real process adversary $\mathcal{A}$ who has no idea he is being simulated. The tally on the tapes of the voters $\hat{V}_i$ corresponds to the inputs given by $\mathcal{E}$ to the voters at the start of the experiment or by $\mathcal{A}$ who has corrupted them. $\mathcal{E}$, like $\mathcal{A}$, cannot break the semantic security of the cryptosystem nor does he have any cause to mistrust the random oracle's output.

Universal verifiability. Except for $\mathcal{F}_{\mathsf{PGK}}$, $\mathcal{F}_{\mathsf{SKG}}$ and $\mathcal{F}_{\mathsf{RO}}$, which are trusted functionalities, all steps are either reproducible by the verifier himself or else accompanied by a zero-knowledge proof. □

B.4. Security and Verifiability of the Authenticated Voting Credential System

PROOF. of Theorem 5.1. *Correctness.* Correctness follows trivially from construction. All voters V_i cast their vote v_i. The tally is given by $t = \sum_i v_i$.

UC-security. We show that the protocol is UC-secure by showing that for every adversary $\mathcal{A}$ that attacks the protocol $\mathcal{P}_{\mathsf{AVS}}$, there is an adversary-simulator $\mathcal{S}$ that attacks the ideal functionality $\mathcal{F}_{\mathsf{VS}}$ with equal success and such that no computationally bounded external environment $\mathcal{E}$ can tell whether he is interacting with $\mathcal{A}$ and $\mathcal{P}_{\mathsf{AVS}}$ or with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{VS}}$.

The adversary-simulator $\mathcal{S}$ runs with dummy voters $\hat{V}_i$, who receive their input from $\mathcal{E}$ and with dummy authorities $\hat{A}_j$. After the experiment is finished, $\mathcal{E}$ looks at the tapes of the voters V_i and authorities A_j, computes for a polynomial duration and eventually outputs a guess as to whether he is interacting with $\mathcal{A}$ or $\mathcal{S}$. Throughout the experiment, $\mathcal{E}$ can communicate with the adversary.

The adversary-simulator simulates an execution of the protocol between simulated authorities A_j, simulated voters V_i and a simulated real-protocol adversary $\mathcal{A}$. The parties A_j or V_i that were corrupted by $\mathcal{A}$ correspond to the parties $\hat{A}_j$ or $\hat{V}_i$ that were corrupted by $\mathcal{S}$. The adversary-simulator proceeds as follows:

— $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{E}$.
— $\mathcal{S}$ starts by simulating the invocation of $\mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{SKG}}$ and simulates $\mathcal{F}_{\mathsf{BB}}$ throughout. As a result, $\mathcal{S}$ knows all private keys and private key shares.
— $\mathcal{S}$ instructs all honest voters V_i to withdraw a credential.
— $\mathcal{S}$ instructs all honest authorities A_i to approve all credential withdrawal requests from eligible voters, including from the corrupted voters (provided that they made a valid credential withdrawal request).
— $\mathcal{S}$ instructs the authorities to make the $\mathcal{F}_{\mathsf{BB}}$ unreadable by sending $(\textbf{close}, sid)$ to the bulletin board.
— If any corrupt V_i casts a vote (i.e.: posts a $(v_{i'}, a, Z)$-message to $\mathcal{F}_{\mathsf{BB}}$), then $\mathcal{S}$ determines who cast the vote and he instructs the matching $\hat{V}_i$ to cast the same vote in $\mathcal{F}_{\mathsf{VS}}$. $\mathcal{S}$ allows this vote to pass.

— If $\mathcal{S}$ receives $(\textbf{vote}, \hat{V}_i, sid)$ from $\mathcal{F}_{\mathsf{VS}}$ pertaining to an honest voter $\hat{V}_i$, $\mathcal{S}$ allows it to pass.
— When all $\hat{V}_i$ have cast their votes, $\mathcal{S}$ allows $\mathcal{F}_{\mathsf{VS}}$ to compute the tally t. From this tally, $\mathcal{S}$ is able to infer the votes that were cast by all the honest voters, without mapping any one vote to the voter that cast it. $\mathcal{S}$ chooses a random mapping of honest votes $v_{i'}$ to honest voters V_i. For each of these honest voters V_i, the bulletin board $\mathcal{F}_{\mathsf{BB}}$ stores a post of the form $(v_{i'}, a, Z)$, where $v_{i'}$ is the vote, a is the preimage of V_i's credential and Z is a non-interactive proof transcript attesting to knowledge of a certificate and involving a and $v_{i'}$. The timestamps of these posts are chosen at random. (The timestamps of the posts from the corrupt parties are not modified.)
— At this point, $\mathcal{S}$ instructs the honest authorities A_i to open $\mathcal{F}_{\mathsf{BB}}$ for reading by sending a $(\textbf{read}, sid)$ message. All participants including $\mathcal{A}$ can read the bulletin board and calculate the tally.

In order to discover that he is being simulated by the adversary-simulator $\mathcal{S}$, the real protocol adversary $\mathcal{A}$ must somehow discover a discrepancy. However, since the adversary cannot observe the behavior of the honest parties except by what they post to the bulletin board, he has nothing to compare the timings of their posts to and hence no indication that suggests they are not authentic. Moreover, the timings of the posts of the corrupted parties conforms to what the adversary instructed them to do.

Since access to the bulletin board is anonymous and since the credentials are unlinkable to the transcript of the protocol that created them, the adversary has no way to determine which of the honest votes was cast by which honest voter.

The external environment $\mathcal{E}$ communicates with $\mathcal{A}$ throughout the protocol. However, $\mathcal{A}$ has no idea he is being simulated. Moreover, there is no indication that there is a $\mathcal{S}$ relaying messages between $\mathcal{E}$ and a simulated $\mathcal{A}$. Since the credentials are unlinkable to the voters, $\mathcal{E}$ cannot compare the mapping between cast votes and the voters of the protocol with the mapping implicit in his input to the voters at the start of the experiment.

Universal verifiability. Vote casting is verifiable as it is accompanied by a zero-knowledge proof which authenticates the credential as well as the vote. It is trivial to check that credentials are not reused as the value a must remain fixed. The verifier can check that only eligible voters are assigned credentials via the bulletin board. This cannot happen as long as at least t authorities are honest. In this sense, verifiability of correctness is still universal (i.e.: *anyone* can verify), but correctness itself is reduced to an assumption of *trust*, down from no assumption at all. □