2012 IEEE Symposium on Security and Privacy

Clash Attacks on the Verifiability of E-Voting Systems

Ralf Küsters, Tomasz Truderung and Andreas Vogt
University of Trier, Germany
{kuesters,truderung,vogt}@uni-trier.de

***Abstract*—Verifiability is a central property of modern e-voting systems. Intuitively, verifiability means that voters can check that their votes were actually counted and that the published result of the election is correct, even if the voting machines/authorities are (partially) untrusted.**

In this paper, we raise awareness of a simple attack, which we call a clash attack, on the verifiability of e-voting systems. The main idea behind this attack is that voting machines manage to provide different voters with the same receipt. As a result, the voting authorities can safely replace ballots by new ballots, and by this, manipulate the election without being detected.

This attack does not seem to have attracted much attention in the literature. Even though the attack is quite simple, we show that, under reasonable trust assumptions, it applies to several e-voting systems that have been designed to provide verifiability. In particular, we show that it applies to the prominent ThreeBallot and VAV voting systems as well as to two e-voting systems that have been deployed in real elections: the Wombat Voting system and a variant of the Helios voting system.

We discuss countermeasures for each of these systems and for (various variants of) Helios provide a formal analysis based on a rigorous definition of verifiability. More precisely, our analysis of Helios is with respect to the more general and in the area of e-voting often overlooked notion of accountability.

***Keywords*-accountability; verifiability; protocol analysis; voting**

I. Introduction

Verifiability is a fundamental property of modern e-voting systems. Intuitively, a verifiable e-voting system allows voters to make sure that their votes were actually counted and that the published result of the election is correct, even if voting machines/authorities are (partially) untrusted. In order to achieve verifiability, voters are typically provided with some kind of receipt which they can use—together with additional data published on a bulletin board, such as encrypted ballots and zero-knowledge proofs—to check that their votes were counted and that voting machines/authorities followed the prescribed procedure.

The main goal of this paper is to raise awareness of a simple attack on verifiability, which does not seem to have attracted much attention in the literature, and to discuss and (formally) analyze countermeasures. The simple idea behind the attack, which we call a *clash attack*, is as follows: Voting machines try to provide different voters with the same receipt—hence, the name of the attack. As a result, authorities can safely replace ballots by new ballots on the bulletin board, and thus, manipulate the election without being detected. We show that, surprisingly, several e-voting systems that have been designed to provide verifiability, among them systems that have been used in real elections, are vulnerable to this attack, under realistic trust assumptions on voting machines and authorities. Our findings illustrate that this attack is a potentially dangerous attack for a large class of e-voting systems. We note that the clash attack can work even if voters and election observers know exactly how many and which voters voted. So clash attacks are different from and more subtle than the well-known ballot stuffing attacks (see, e.g., [4] for ballot stuffing attacks).

More precisely, the contribution of this paper is as follows.

Contribution of this Paper. We show that the clash attack applies to the prominent voting systems ThreeBallot and VAV [12], the Wombat voting system [16], which was used in an election at an Israeli college, and variants of the Helios voting system [1]. In the variant of Helios where each ballot (basically an encrypted vote) is published on the bulletin board next to the name of the voter who cast the ballot, the clash attack does not work (see below); we refer to this variant as the *original variant of Helios*. However, this variant is not an option for many elections. One reason is that the mere fact of voting is often required to be confidential by law, see e.g., [2]. Moreover, it might be possible to break the encryption of votes in the future, which would then reveal how specific voters voted. Therefore, another variant of Helios was proposed [2] where voters are provided with aliases (the *variant of Helios with aliases*). In this variant, ballots are published on the bulletin board along with the aliases, instead of the names of voters. This variant of Helios has been used for a presidential election at a Belgian university [2] and also for the IACR renewal of the board of directors election [5]. We show that for this variant of Helios the clash attack works, under the realistic assumption that the assignment of aliases to voters is not trusted. Another natural variant of Helios is to merely publish the ballots without voter names/aliases or to publish the ballots detached from the voter names/aliases (the *variant of Helios with detached names*). We show that for this variant of Helios too the clash attack can be carried out successfully.

We discuss several countermeasures for the discovered clash attacks. Since Helios is probably the most prominent and practical e-voting system among the ones we have studied, we also carry out formal analysis for the different variants of Helios (with our countermeasures applied, if necessary): the original variant, the variant with aliases, and the variant

DOI 10.1109/SP.2012.32

with detached names. More specifically, we do not only study the verifiability property for Helios but also the accountability property. This analysis is of independent interest since so far verifiability has only very rarely been formally analyzed for concrete e-voting systems [8], [14], [6], [15], [10] and accountability has almost never been studied (formally or informally) for concrete e-voting systems, with the formal analysis of the Bingo voting system carried in [10] being an exception.

Rigorous cryptographic definitions of accountability and verifiability were introduced by Küsters et al. in [10]. Our analysis uses these definitions. Intuitively, in the context of voting, accountability means that if the published outcome of the election is not correct, i.e., does not correspond to how the voters actually voted, and hence, some party must have misbehaved (intentionally or unintentionally), then someone—ideally (a group of) voting authorities—can be held accountable for the misbehavior. It was shown in [10] that verifiability can be seen as a special case of accountability. A very useful feature of the definitions by Küsters et al. is that they permit to precisely measure the level of accountability (verifiability) an e-voting system provides: it can be precisely captured under which conditions which (group of) parties can be held accountable and how likely it is that misbehavior goes unnoticed. Our formal analysis of Helios shows that we obtain different levels of accountability for the three variants of Helios—the original variant, the variant with aliases, and the variant with detached names.

Orthogonally to the clash attack, our formal analysis also reveals some quite obvious, but serious problem with the accountability of the Helios system: If a voter claims that her ballot is not published or not correctly displayed on the bulletin board, then it is unclear whether the voter misbehaved, e.g., simply lied, or the authority/bulletin board tried to cheat. Fixing this problem would probably require a major redesign of the system.

Related Work. As mentioned, our formal analysis of Helios uses the definitions of accountability and verifiability proposed by Küsters et al. [10]. Their definition of verifiability takes a global view on verifiability, unlike the traditional notions of individual and universal verifiability. It was recently demonstrated in [9] by attacks on the verifiability of ThreeBallot and VAV [12] that individual and universal verifiability in general do not imply (global) verifiability. We note that the attacks found in [9] use specific details of ThreeBallot and VAV and are very different from the clash attack considered here, which applies to a much broader class of e-voting systems (see also Section IV-C).

The clash attack should not be confused with the so-called trash attack [3]. The idea of the trash attack is that if voters throw away their (paper) receipt, then authorities who find these receipts could conclude that these voters will not check their receipts on the bulletin board, and hence, ballots of such voters can safely be modified. In contrast, the clash attack also works if all voters check their receipts.

Helios was analyzed by Kremer et al. [8] with respect to verifiability in a symbolic Dolev-Yao style model, rather than a cryptographic model; accountability has not been studied for Helios so far. The definition of verifiability and the model of Helios Kremer et al. use is quite coarse. For example, their definition of verifiability assumes that all voters are honest. (The attack in [9] demonstrates that the existence of dishonest voters may introduce new attacks.). Also, unlike the definition in [10], their definition does not allow them to measure the level of verifiability, i.e., to calculate the probability that manipulations go undetected, and to precisely determine under what circumstances it is guaranteed that the election can be carried out without complaints. While their definition of verifiability in principle captures clash attacks, Helios is modeled by Kremer et al. in such a way that clash attacks are excluded and these attacks are not discussed in their paper. We also note that Kremer et al. do not model the auditing of ballots by voters. In their model, ballots are simply guaranteed to contain the vote intended by the voter; in fact, since Kremer et al. have a possibilistic model, faithfully modeling the audit process would be rather tricky.

A logic-based formulation of accountability was presented in [7], which, however, does not seem to have been applied to a concrete protocol.

Structure of the Paper. In the following three sections, the clash attacks on the Wombat voting system, the variants of Helios as well as ThreeBallot and VAV are discussed informally. For the formal treatment of Helios, in Section V we first recall the definitions of accountability and verifiability proposed by Küsters et al. in [10]. We then formally analyze accountability and verifiability of the different variants of Helios in the subsequent sections. We conclude in Section IX. Some more details are provided in the appendix; see our technical report [11] for full details.

II. Wombat Voting

In this section, we briefly recall the Wombat voting system, present the clash attack on this system, and sketch some countermeasures.

A. The System

The Wombat voting system was recently developed under the lead of Alon Rosen, Amnon Ta-shma, Ben Riva, and Jonathan Ben-Nun and was used in an election at the Interdisciplinay Center (IDC) in Herzliya, Israel [16], in which the students union chairman and vice-chairman as well as the director of a school within IDC were elected. It is a voting booth scheme which consists of the following steps:

1. A voter presents her id-card at the polling station, where a number of clerks check that the voter is eligible and has not voted so far.
2. The voter enters the voting booth and indicates her choice by pressing the name of her candidate on the touchscreen of the voting machine. The voting machine then prints a ballot consisting of the chosen candidate's name (in plaintext) and the encryption of this name (with some threshold public key scheme), together with a (random-looking) serial number.

3. The voter can now either opt for casting this ballot or for auditing it. If she chooses to audit, the machine supplies the random coins used in the encryption, which allows the voter (or anyone else)[1] to check that the previously printed encryption is indeed an encryption of the chosen candidate. This audited ballot is not valid anymore and the voter can vote again.
4. If eventually the voter decides to cast a ballot, she detaches the candidate's name (in plaintext) from the part containing the encryption and the serial number. The latter part is scanned by the clerks and published on a bulletin board. The part of the ballot with the candidate's name in clear is put into a separate box. This paper trail can, if inconsistencies occur, be used for manual recounting. However, as we will see later, our attack will not cause any observable inconsistencies; therefore, there will be no reason for a manual recount.
5. The part with the encrypted ballot and the random-looking serial number serves as a receipt for the voter. Having this data, the voter can check that her ballot indeed appears on the bulletin board.
6. When the voting phase is finished, the data on the bulletin board is first sent through a mixnet and then decrypted (in a threshold manner) by a group of trustees.

B. The Clash Attack on Wombat Voting

The authors of this system say that, under the (reasonable) assumption that one trusts the clerks to only let eligible voters vote once, the integrity of the system is *always* preserved even in the case "where a hacker gets full control of both the software and all the secret keys of the system [...] Thus, if the elections pass audit and are successfully verified by voters, then voters can be assured that the election results are correct.".[2] This, however, is not true! The Wombat scheme is vulnerable to the clash attack if the voting machine and the bulletin board collaborate (which definitely is a scenario in the scope of "a hacker gets full control of the system"), even if all voters check their receipts and all published data, such as zero-knowledge proofs, are successfully verified.

The clash attack on the Wombat system works as follows: In the extreme case, the voting machine uses, for every ballot, the same random coin r for encrypting a vote and, for every candidate c, the same serial number s_c in all the ballots that contain a vote for c. As a result, voters who voted for the same candidate c will get the same receipt, namely a receipt containing the ciphertext $Enc_{pk}(c,r)$ and the serial number s_c. Now the bulletin board, for every candidate c, can leave only one original ballot, and safely replace all the remaining ones by ballots of its choice. Note that every voter will find "her" ballot on the bulletin board, as one copy of every ballot produced in the voting phase is kept. So, the bulletin board (although manipulated) appears to be consistent and the remaining phase—the tallying of the ballots as they appear on the bulletin board—can follow the prescribed procedure. Note that the attack works even if everybody knows exactly how many and which voters voted.

The general observation is that for every duplicated receipt, produced by the voting machine, the bulletin board can safely manipulate one vote. So the attack is successful even if the voting machine is more subtle than described above and uses many more random coins and serial numbers.

We emphasize that in the current procedure clerks and voters are not obligated to detect duplicated receipts. Also, for humans, noticing duplicates is not easy anyway in the Wombat system since an encrypted vote is presented as a QR code (a two dimensional bar code) and the serial numbers are, as mentioned, random-looking. Hence, without writing down serial numbers, it is quite hard for voters and clerks to notice duplicates. In particular, clerks will not get suspicious if the voting machine produces duplicated receipts only from time to time. Moreover, the voting machine can easily prevent that a single voter sees duplicated receipts by using new serial numbers and new randomness for the cast ballot and the one audited by the voter. (Note that the clash attack will nevertheless be very effective.)

We also note that for the clash attack to work, it is not necessary that the voting machine is completely corrupted. It is sufficient if the part of the machine that is responsible for the random number generation is flawed (and produces some number of clashes).

As mentioned before, the Wombat voting system keeps a paper trail. So if the paper ballots were recounted, the clash attack would be detected. However, the point of a verifiable voting system is to make the recounting superfluous in cases were there have been no complaints. The clash attack for the Wombat system shows that it is possible to manipulate the outcome of the election without risking any complaints.

C. Countermeasures

The above scheme can be fixed in several ways in order to prevent our attack. One possible fix is to print serial numbers on receipts in advance, instead of letting the voting machine print these numbers, and in case one does not trust the process of printing the serial numbers, check that no number is printed twice. Another fix is to let the clerks collect all serial numbers of ballots that were cast and check whether duplicates occur. Assuming that the clerks are trusted (or there is a least one honest clerk among the group of clerks), this again prevents the clash attack, or more precisely, the clash attack would be detected by the clerks.

Similarly, voters might compare their receipts and, by this, prevent the clash attack; how this is done exactly, should however be made explicit in the voting procedure. We note that publishing/comparing receipts naively may have negative consequences regarding privacy, analogously to a setting where receipts are published along with the names of voters (see Section I).

A formal analysis of the Wombat voting system with these fixes applied would be very similar to the formal analysis we

[1]The authors of the Wombat system implemented an Android app for that purpose, available at https://market.android.com/details?id=com.veriballot.

[2]http://www.wombat-voting.com/faq

carry out for the variants of Helios in this paper. Since the Helios voting scheme is more influential, we only focus on the formal analysis of Helios in this paper.

III. Helios

We now recall several variants of the Helios voting system, present the clash attacks on these systems, as far as applicable, and discuss countermeasures. A formal analysis of Helios (with the countermeasures applied) is carried out in the subsequent sections.

A. The System

There are different variants of Helios and still new versions of Helios are proposed. However, all these variants have the following structure:

1. Voters can use their browsers to create and submit ballots. More precisely, the browser (or, more precisely, the voter's client program run in the browser) encrypts the choice of a voter, resulting in a ballot; if a variant of Helios with homomorphic encryption is used, the ballot also contains a zero-knowledge proof testifying the well-formedness of the ballot. A voter can then opt for casting that ballot or for auditing it. If the voter chooses to audit the ballot, the browser provides the voter with the used randomness. The voter is then supposed to check, with the help of a trusted tool, that the browser in fact encrypted the voter's choice, i.e., whether the voter's choice encrypted using the randomness provided by the browser indeed yields the audited ballot. Audited ballots cannot be cast. Voters are therefore asked to vote again.
2. Once a voter has decided to cast the ballot, she is asked to authenticate herself using some credential. The authentication can be done directly on the voting server or on an intermediate authentication server, which, after a successful authentication, passes the ballot to the voting server.
3. The ballots are published on a bulletin board after the voting phase is finished. (Below we also comment on the case were ballots are visible on the bulletin board immediately.)
4. The ballots are now first sent through a verifiable mixnet and then decrypted by a set of trustees, or, if an homomorphic encryption scheme is used, the ballots are first multiplied and then the resulting ciphertext is decrypted.

We mention that in Helios, typically the voters may revote, that is, after having submitted a ballot, at any point before the voting phase is finished, a voter may submit a new ballot which replaces the old one. However, for the ease of presentation, we assume that voters do not revote. We emphasize that our results—both positive and negative ones—still hold true without this simplification.

The variants of Helios differ, besides other things that are not relevant for the clash attack, in the kind of information posted on the bulletin board:

The original variant: In the original variant [1], the name of a voter is published next to the voter's ballot. This variant is immune to our attack (see Section VI for a formal analysis). However, as already mentioned in the introduction, this variant of Helios has several disadvantages: (a) everybody learns who abstained from voting, information that in some cases must not be revealed by law, see, e.g., [2], and (b) it is possible that in the future, the encryption scheme used to encrypt votes can be broken, and hence, votes of individual voters would be revealed.

Helios with aliases: In a variant of Helios proposed in [2] and used in two real elections [5], [2], voters obtain aliases from the election authority in a registration phase. In this variant ballots of voters are published on the bulletin board next to these aliases, instead of the actual names of the voters. Besides being additional work, the issuing of aliases is security relevant, as will become clear in Section III-B.

Helios with detached names: Other natural variants of Helios without the disadvantages mentioned above are that the names of voters are not published at all on the bulletin board or that the names are published in lexicographic order, detached from the ballots. The former variant has the advantage that nobody learns who abstained from voting, whereas in the latter variant ballot stuffing (i.e., adding a ballot for every voter that did not vote) is somewhat more difficult for voting authorities. However, both variants are vulnerable to the clash attack, as discussed next.

B. Clash Attacks on the Variants of Helios with Aliases and Detached Names

We now show how the clash attack can be mounted on the variant of Helios with aliases and the variant with detached names as described in Section III-A; as mentioned, the original variant of Helios is immune to the clash attack. Note that Helios shares a lot of ideas with the Wombat voting system as far as the clash attack is concerned. The main difference is that in Helios, instead of a voting machine, voters use their browsers to cast ballots.

Helios with Detached Names. For the clash attack to work for this variant of Helios, we assume that the browser (i.e. the client program run in the browser) and the bulletin board are dishonest; the tallying will be done correctly. Now, the browser works as follows for every voter. It always uses the same sequence $r_1, r_2, \dots$ of random coins: r_1 is used to produce the first ballot (i.e. the first ballot contains an encrypted vote of the form $Enc_{pk}(c, r_1)$, where c is the candidate chosen by the voter), r_2 is used to produce the second ballot, and so on.[3]

As a result, many identical ballots reach the bulletin board. More precisely, all the voters who vote for the same candidate and have audited their ballots the same number of times cast identical ballots and obtain identical receipts. The bulletin board can then safely replace all but one of each of the duplicates by arbitrary ballots (i.e., encryptions of candidates of its choice), and hence, safely manipulate the result of the election. In fact, nobody will detect any manipulation: every

[3]If homomorphic encryption is used, the random coins r_i are also used to procedure the mentioned zero-knowledge proofs.

voter who cast her ballot can check that her name, in the case where names are published (detached from ballots), as well as her ballot appears on the bulletin board. Note that the attack works even if voters know exactly how many and which voters voted.

Helios with Aliases. Here we assume that the browser (i.e., the client program run in the browser), the bulletin board, and the authority in charge of issuing aliases to voters are dishonest; the tallying will again be done correctly.

It is realistic to assume that an adversary knows for sure for some voters that they will vote and how they will vote, provided that the voters do not have too many choices. For example, very active members of a political party will not abstain from voting and vote for the same political party, namely their party. Now, the idea behind the clash attack for Helios with aliases is that the authority in charge of issuing aliases to voters could issue the same alias to two or more voters for which the authority knows that they will make the same choice, i.e., vote for the same candidate/party. All browsers, or at least those for voters with the same aliases, always use the same randomness for preparing the ballots. By this, a voter who obtained the same alias as another voter will obtain the same receipt, and again, the bulletin board can replace all but one of those duplicates by arbitrary ballots. Provided that those voters who were issued the same alias in fact vote for the same candidate/party (the adversary would not risk to issue the same alias if he could not be sure of that), no manipulation will be detected: every voter who cast a ballot can check that the ballot next to "her" alias coincides with "her" receipt. Again, we note that the attack works even if voters know exactly how many and which voters voted.

An important difference to the clash attack for Helios with detached names is that now the browser always uses the same random coins (at least for those voters who obtained the same alias). If the browser used a sequence $r_1, r_2, \ldots$ of random coins to encrypt consecutive choices of voters, as in the case of Helios with detached names, the attack would very likely be detected: If two voters with the same alias, say a, choose to audit their ballots a different number of times, then two different ballots, say b_1 and b_2, are produced for them. Now, the bulletin board can only publish one of these ballots on the bulletin board along with the alias a. (There should be at most one entry on the bulletin board for every alias.) But then, one of the voters with alias a could detect a mismatch.

This is why in the clash attack on Helios with aliases, the *same* random coins r are used to encrypt all the votes for voters who share the same alias. Consequently, all the ballots produced for such a voter are the same. Note that the audit procedure, as proposed in [2], only mandates to check whether the audited ballot is indeed an encryption of the chosen candidate. It does not required to check whether the ballots produced in the consecutive steps are different (use different random coins). Therefore, as long as this audit procedure is carried out, no manipulation is detected. Also note that voters are not asked to remember the ballots produced in different steps and that these ballots (or their fingerprint) are quite hard to remember since their representations are rather long strings. Voters will therefore typically not remember these strings but simply copy and paste them for the purpose of auditing and otherwise forget about them.

We finally note that the clash attack is more risky to carry out, if ballots (along with the aliases) are published on the bulletin board not after the voting phase is finished but right away—which also means that the election turnout is published before the end of the election phase. For example, this is done in the IACR election [5]: A voter with alias a might check the bulletin board before she votes by herself and then might detect that the bulletin board already contains an entry for a, namely one generated by another voter with the same alias.

C. Countermeasures

Helios with Detached Names. To prevent the clash attack on Helios with detached names, we let the voter contribute to the randomness that is used for encryption. One way of implementing this is to ask the voter to type in a "random" string before the ballot is produced by the browser. The voter-provided randomness is then combined with the random coins generated by the browser using a collision resistant hash function h.[4] More specifically, the voting procedure consists of the following steps:

1. The voter indicates her choice c and provides a random string r_v (similarly to a one-time password). For simplicity of the argument, we assume such random strings to all have the same length (e.g., eight characters).
2. The browser chooses a random coin r_b, then computes $r = h(r_b||r_v)$, and uses r to encrypt the voter's choice, where $r_b||r_v$ denotes the concatenation of r_b and r_v.
3. Now, if the voter chooses to audit the ballot, the browser is supposed to provide its random coin r_b. This allows the voter (by using some independent program) to check whether the ballot has been produced correctly: the voter recalculates the encryption of her choice using r_b and the random string r_v provided by herself.

Intuitively, the above procedure prevents the clash attack (or at least makes it very risky for the adversary) because it is quite unlikely that two voters who choose the same candidate also choose the same voter-provided random string for the ballot cast. Note that due to the collision resistance of h it is computationally infeasible for the browser, given two different voter-provided random strings r_v and r'_v of the same length to produce random coins r_b and r'_b such that $h(r_b||r_v) = h(r'_b||r'_v)$.

In Section VII we formally show that this variant of Helios provides a reasonable level of accountability and verifiability. (We also formally show that this variant of Helios without the fix does not provide a sufficient level of accountability and verifiability.) Clearly, the level of security achieved depends on the entropy of the voter-provided randomness.

[4]To prove vote privacy of this variant of Helios, a stronger assumption on h is necessary. Basically, h has to produce random strings when given random strings as input, which is, for instance, the case when h is modeled as a random oracle.

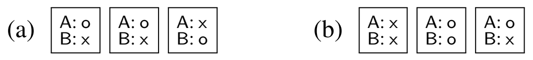

Fig. 1. Two ways of voting for the second candidate (candidate B) in the ThreeBallot protocol, where x represents a marked position and o represents an unmarked position. All the other possibilities of voting for B can be obtained as permutations of these two.

Helios with Aliases. One way of preventing the clash attack is to apply the same countermeasure as the one described above for Helios with detached names. Another countermeasure, which would not work for Helios with detached names, is the following:

As discussed, for the clash attack on Helios with aliases to be effective, a browser has to always use the same randomness (for each ballot of a given voter), which then leads to identical ballots. A natural countermeasure is therefore to modify the audit procedure voters carry out as follows: In addition to checking whether the audited ballot in fact is computed with the voter's choice and the random coins provided by the browser for that ballot, a voter also makes sure that all ballots (including the submitted ballot) constructed by the browser are different. For this purpose, a voter would typically record the ballots in some way, e.g., by having them emailed to her, and then compare these ballots.

The intuition behind this countermeasure is that the browser now has to use fresh ballots in every round. Now, if two voters are assigned the same alias, they may choose to audit their ballots a different number of times and, by the above, submit different ballots with the same alias. Hence, the clash attack would be detected with non-negligible probability.

We prove in Section VIII that this variant of Helios in fact has a relatively good level of accountability and verifiability. Conversely, we also formally show that this variant of Helios without the fix does not provide a sufficient level of accountability and verifiability.

IV. ThreeBallot and VAV

We now briefly recall the ThreeBallot and VAV voting systems [13], [12], following the representation in [9]. Then we discuss how the clash attack applies to these systems.

A. *The ThreeBallot System*

In ThreeBallot [12], a voter is given a multi-ballot consisting of three simple ballots, also called a *three-ballot*. On every simple ballot the candidates are printed in the same fixed order. In the secrecy of a voting booth, the voter is supposed to fill out all three simple ballots in the following way: She marks the candidate of her choice on exactly *two* simple ballots and every other candidate on exactly *one* simple ballot; Figure 1 shows two ways of voting for candidate B in an election with two candidates, candidate A and candidate B. After this, she feeds all three simple ballots to a voting machine (some kind of scanner) and indicates the simple ballot she wants to get as a receipt. The machine checks the well-formedness of the three-ballot, prints secretly random numbers on each simple ballot, where numbers on different simple ballots are chosen independently, and gives the voter a copy of the chosen simple ballot, with the random number printed on it. Note that the voter does not get to see the random numbers of the remaining two simple ballots. As usual for ThreeBallot, we assume that the machine is constructed in such a way that it does not get to know which simple ballot has been taken as a receipt. The scanner keeps all three simple ballots (now separated) in a ballot box. We assume that clerks guarantee that only registered voters can vote and that every voter votes at most once.

In the tallying phase, the voting machine posts all (electronic copies of) the cast simple ballots in a random order on the bulletin board. From the ballots shown on the bulletin board the result can easily be computed: the number of votes for the i-th candidate is the number of simple ballots with the i-th position marked minus the total number of votes, which is the total number of simple ballots on the bulletin board divided by three.

ThreeBallot was meant to provide (some level of) verifiability. A crucial assumption for this, already made in the original paper [12], is that neither the scanner, the voting authority, nor the bulletin board know which simple ballots have been taken as receipts by honest voters before all ballots are published on the bulletin board. Now the intuition behind why ThreeBallot provides (some level of) verifiability is that it should be risky for any party to remove or alter simple ballots in order to manipulate the result since the probability that the modification of k simple ballots goes undetected is merely $(\frac{2}{3})^k$.

Unfortunately, as we will see in Section IV-C, this argument, found in the literature, is flawed.

B. *The VAV System*

In VAV [12], a voter is given a *three-ballot* consisting of three simple ballots. On every simple ballot the candidates are printed in the same fixed order. On the top of one of those simple ballots the letter A is printed; on the top of the remaining two simple ballots the letter V is printed. In the secrecy of a voting booth, the voter is supposed to fill out her three-ballot in the following way: (S1) She marks the position next to the candidate of her choice on one of the V-ballots and then (S2) she marks the position next to some randomly chosen candidate on the two remaining simple ballots, one V- and one A-ballot. Figure 2 shows all three ways of filling out the three-ballot for candidate 1 in an election with three candidates. After this, she feeds all three simple ballots to a voting machine (some kind of scanner) and indicates the simple ballot she wants to get as a receipt. The machine checks the well-formedness of the three-ballot, prints secretly random numbers on each simple ballot, where numbers on different simple ballots are chosen independently, and gives the voter a copy of the chosen simple ballot, with the random number printed on it. Note that the voter does not get to see the random numbers of the remaining two simple ballots. The scanner keeps all simple ballots (now separated) in a ballot box.

In the tallying phase, the voting machine posts all the (electronic copies of) cast simple ballots in a random order

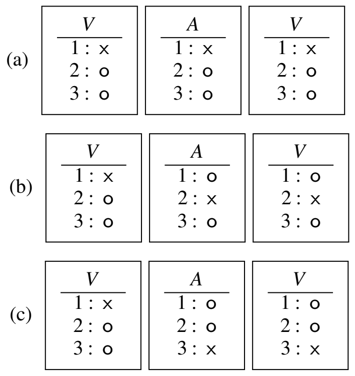

Fig. 2. Three ways of voting for the candidate 1 in the VAV protocol, where x represents a marked position and o represents an unmarked position.

on the bulletin board. From the ballots shown on the bulletin board the result can easily be computed: All A-ballots and the corresponding V-ballots (i.e. V-ballots marked at the same position) are removed. From the remaining V-ballots the number of votes for each candidate can directly be read off.

Similarly to ThreeBallot, VAV is supposed to provide (some level of) verifiability because it should be risky for any party to remove or alter simple ballots in order to manipulate the result, again under the assumption that neither the scanner, the voting authority, nor the bulletin board know which simple ballots have been taken as receipts by honest voters before all ballots are published on the bulletin board.

C. The Clash Attack on ThreeBallot and VAV

We first present the clash attack on ThreeBallot. We assume that the voting machine (i.e., the scanner) and the bulletin board are dishonest.

The idea of the attack is as follows: The voting machine simply prints the same serial number on those simple ballots which have the same pattern, i.e., where the same candidates are marked. We note that it is not necessary that all such simple ballots get the same serial number. It suffices if some get the same number. (Roughly speaking, the number of clashes corresponds to the number of votes that can safely be manipulated.) The bulletin board can now safely replace all but one simple ballot with the same serial number by simple ballots of its choice (with new serial numbers). The bulletin board only needs to make sure that altogether the published set of simple ballots remains consistent, i.e., the set of simple ballots can be grouped into a set of well-formed three-ballots, and hence, votes. This can in fact easily be done as follows: First, it is not hard to see that a given consistent set of simple ballots can efficiently be grouped into a set of consistent three-ballots. Now, for example, if we have two three-ballots of the form

$$\boxed{\begin{smallmatrix}\text{A: x}\\\text{B: x}\end{smallmatrix}}\boxed{\begin{smallmatrix}\text{A: o}\\\text{B: x}\end{smallmatrix}}\boxed{\begin{smallmatrix}\text{A: o}\\\text{B: o}\end{smallmatrix}} \quad \text{and} \quad \boxed{\begin{smallmatrix}\text{A: x}\\\text{B: o}\end{smallmatrix}}\boxed{\begin{smallmatrix}\text{A: o}\\\text{B: x}\end{smallmatrix}}\boxed{\begin{smallmatrix}\text{A: x}\\\text{B: o}\end{smallmatrix}}$$

in that set and the simple ballots in the middle of these two three-ballots have the same serial number, then the bulletin board can replace the simple ballot of the first three-ballot, say, by $\boxed{\begin{smallmatrix}\text{A: x}\\\text{B: o}\end{smallmatrix}}$, which turns the first three-ballot from a vote for B into a vote for A; similarly for other cases.

Since the bulletin board stays consistent and all voters find "their" receipt on the bulletin board, the clash attack goes unnoticed. We emphasize that this attack works even if the scanner and the bulletin board do not know which simple ballots voters chose as receipts. Also note that clerks in polling stations do not necessarily get to see simple ballots with the serial numbers printed on them and voters typically only get to see their receipts; occasionally maybe a few more receipts of other voters. So it is rather unlikely that duplicates will be noticed, and moreover, voters and clerks are not required to check for duplicates.

Similarly to the Wombat voting system, ThreeBallot keeps a paper trail and recounting of ballots would uncover the clash attack. But again, the point of a verifiable voting system is to make the recounting superfluous in cases where there have been no complaints.

Given the clash attack on ThreeBallot presented above, it is easy to see how the clash attack on VAV works. The main idea is the same: The scanner prints the same serial number on (at least some) simple ballots with the same pattern. The bulletin board can then safely modify duplicated simple ballots and by this manipulate the result of the election.

As already mentioned in the introduction, in [9] another attack on the verifiability of ThreeBallot and VAV was presented. This attack assumes a dishonest bulletin board and a dishonest voter (or alternatively a dishonest scanner). The main idea here is that the dishonest voter tells the bulletin board the serial number of its receipt. The bulletin board can then change this voters receipt on the bulletin board, which altogether corresponds to a voter casting an invalid three-ballot, say one where on all three simple ballots candidate B is marked. Now, the trick is that this invalid three-ballot together with a three-ballot which represents a vote for A form two votes for B. So, the attack presented in [9] and the one presented here are quite different: the trust assumptions are different and the basic ideas are different, in particular, clashes do not play a role in [9].

One countermeasure for the clash attack, which is similar to one of the proposals for the Wombat system, would be to print serial numbers on ballots in advance and possibly conceal them (with a scratch strip) or, alternatively, let a different machine, which does not learn the patterns, print the serial numbers. However, since ThreeBallot and VAV have other quite serious problems with verifiability [9], we do not elaborate further on countermeasures for these systems.

V. Accountability and Verifiability

In this section, we briefly recall the definition of (computational) accountability and verifiability of [10]. We start with the notion of a protocol. For the purpose of this paper, a protocol describes a voting system, including the honest and possible dishonest behavior of parties.

A. Protocols

A *protocol* P specifies the set of agents and channels these agents can communicate over. Moreover, P specifies, for every agent a, a set Π_a of all programs a can run and a set $\hat{\Pi}_a \subseteq \Pi_a$ of *honest programs of* a, i.e., the set of programs that a runs if a is honest. (Formally, a program is modeled as a probabilistic polynomial-time interactive Turing machine.)

Let P be a protocol with agents $a_1,\ldots,a_n$. An *instance of* P is a process of the form $\pi = (\pi_{\mathsf{a}_1} \parallel \ldots \parallel \pi_{\mathsf{a}_n})$ with $\pi_{\mathsf{a}_i} \in \Pi_{a_i}$, where $\parallel$ denotes process composition. An agent a_i is *honest* in such an instance, if $\pi_{\mathsf{a}_i} \in \hat{\Pi}_{\mathsf{a}_i}$. A *run of* P is a run of some instance of P. An agent a_i is honest in a run r, if r is a run of an instance of P with honest a_i. A *property* γ *of* P is a subset of the set of all runs of P. By $\neg\gamma$ we denote the complement of γ.

B. Accountability

The definition of accountability is w.r.t. an agent J of the protocol who is supposed to blame protocol participants in case of misbehavior. The agent J, sometimes referred to as a *judge*, can be a "regular" protocol participant or an (external) judge, who is typically provided with additional information by other, possibly untrusted protocol participants.

Informally speaking, accountability requires two conditions to be satisfied:

(i) (*fairness*) J (almost) never blames protocol participants who are honest, i.e., run their honest program.

(ii) (*completeness*) If, in a run, some desired goal of the protocol is not met—due to the misbehavior of one or more protocol participants—then J blames those participants who misbehaved, or at least some of them (see below).

For example, for voting protocols a desired goal could be that the published result of the election corresponds to the actual votes cast by the voters. The completeness condition then guarantees that if in a run of the protocol the published result of the election does not correspond to the actual votes cast by the voters (a fact that must be due to the misbehavior of one or more protocol participants), then one or more participants are held accountable by J; by the fairness condition they are *rightly* held accountable.

To specify the completeness condition in a fine-grained way, the notions of verdict and an accountability property are used.

A *verdict* is a positive boolean formula ψ built from propositions of the form $\mathsf{dis}(a)$, for an agent a, where $\mathsf{dis}(a)$ is intended to express that a misbehaved (behaved dishonestly), i.e., did not follow the prescribed protocol. For example, in a voting protocol with a voting machine M and auditors $A_1,\ldots,A_r$, a verdict (of a judge) of the form $\mathsf{dis}(M) \vee (\mathsf{dis}(A_1) \wedge \ldots \wedge \mathsf{dis}(A_r))$ expresses the judge's belief that either the voting machine misbehaved or all the auditors misbehaved. We will denote by $\mathscr{F}_{\mathsf{dis}}$ the set of all verdicts.

The agent J, i.e., the judge, can *state a verdict* ψ by sending ψ on its dedicated output channel $\mathsf{decision}_J$. For a protocol P and an instance π of P, a verdict ψ *is true in* π, written $\pi \models \psi$, iff the formula ψ evaluates to true with the proposition $\mathsf{dis}(a)$ set to false, if a is honest in π, and set to true otherwise.

An *accountability constraint* is a tuple $(\alpha, \psi_1, \ldots, \psi_k)$, written $(\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_k)$, where α is a property of P and $\psi_1, \ldots, \psi_k \in \mathscr{F}_{\mathsf{dis}}$. Such a constraint *covers* a run r, if $r \in \alpha$.

Intuitively, in a constraint $C = (\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_k)$ the set α contains runs in which some desired goal of the protocol is *not* met (due to the misbehavior of some protocol participant). The formulas $\psi_1, \ldots, \psi_k$ are the possible (minimal) verdicts that are supposed to be stated by J in such a case; J is free to state stronger verdicts (by the fairness condition these verdicts will be true). Formally, for a run r, J *ensures* C *in* r, if either $r \notin \alpha$ or J states in r a verdict ψ that implies one of $\psi_1, \ldots, \psi_k$ (in the sense of propositional logic).

In practice, so-called *individual accountability* is highly desirable in order to deter parties from misbehaving. Formally, $(\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_k)$ provides *individual accountability*, if for every $i \in \{1, \ldots, k\}$, there exists a party a such that ψ_i implies $\mathsf{dis}(a)$. In other words, each $\psi_1, \ldots, \psi_k$ determines at least one misbehaving party.

A set Φ of constraints for protocol P is called an *accountability property* of P. Typically, an accountability property Φ covers all relevant cases in which desired goals for P are not met, i.e., whenever some desired goal of P is not satisfied in a given run r due to some misbehavior of some protocol participant, then there exists a constraint in Φ which covers r.

As usual, a function f from the natural numbers to the interval $[0,1]$ is *negligible* if, for every $c > 0$, there exists ℓ_0 such that $f(\ell) \leq \frac{1}{\ell^c}$, for all $\ell > \ell_0$. The function f is *overwhelming* if the function $1-f$ is negligible. A function f is δ*-bounded* if, for every $c > 0$ there exists ℓ_0 such that $f(\ell) \leq \delta + \frac{1}{\ell^c}$, for all $\ell > \ell_0$.

Let P be a protocol with the set Σ of agents and Φ be an accountability property of P. Let π be an instance of P, and $J \in \Sigma$ be an agent of P. We write $\Pr[\pi(1^\ell) \mapsto \neg(J:\Phi)]$ to denote the probability that π, with security parameter 1^ℓ, produces a run such that J does not ensure C in this run, for some $C \in \Phi$.

An agent J is *computationally fair* in P, if, for all instances π of P, J states false verdicts only with negligible probability, where a verdict ψ is false in a run r of π if $\pi \not\models \psi$.

Definition 1 (**Computational accountability** [10]). Let P be a protocol with the set of agents Σ, $J \in \Sigma$, an accountability property Φ of P, and $\delta \in [0,1]$. We say that J *ensures* (Φ,δ)*-accountability for protocol* P (or P *is* (Φ,δ)*-accountable w.r.t.* J) if

(i) (*fairness*) J is computationally fair in P and

(ii) (*completeness*) for every instance π of P, the probability $\Pr\left[\pi(1^\ell) \mapsto \neg(J:\Phi)\right]$ is δ-bounded as a function of ℓ.

In the completeness condition, it is of course desirable that $\delta = 0$, i.e., the probability that J fails to ensure a constraint is negligible. However, this is typically too demanding, as illustrated in [10] and by our formal analysis of Helios presented in the subsequent sections.

C. Verifiability

Let P be a protocol and γ be a property of P, called the *goal of P*. An agent J *accepts a run r*, if in this run J sends the message accept on channel $\mathsf{decision}_J$. Intuitively, J accepts a run if she believes that the goal has been achieved in this run. The agent J may be a regular protocol participant, e.g., a voter, or an external judge, who is provided with information by (possibly untrusted) protocol participants.

The definition of verifiability, besides the goal, has an additional parameter: a positive boolean formula over propositions of the form $\mathsf{hon}(a)$, for an agent a. Such a formula describes a group or groups of participants that can guarantee, when running their honest programs, that the goal of the protocol is achieved. We will denote the set of such formulas by $\mathscr{F}_{\mathsf{hon}}$. For example, for a voting protocol with a voting machine M and auditors $A_1,\dots,A_r$, one might expect that to achieve the goal of the protocol it is sufficient that M is honest and at least one of the auditors $A_1,\dots,A_r$ is honest. This can be expressed by the formula $\varphi_{ex} = \mathsf{hon}(M) \wedge (\mathsf{hon}(A_1) \vee \dots \vee \mathsf{hon}(A_r))$.

For an instance π of P and $\psi \in \mathscr{F}_{\mathsf{hon}}$, we write $\pi \models \psi$ if ψ evaluates to true with the proposition $\mathsf{hon}(a)$ set to true, if a is honest in π, and set to false otherwise. By $\Pr[\pi(1^\ell) \mapsto (J : \mathsf{accept})]$ we denote the probability that π, with security parameter 1^ℓ, produces a run which is accepted by J. Analogously, by $\Pr[\pi(1^\ell) \mapsto \neg\gamma,\ (J : \mathsf{accept})]$ we denote the probability that π, with security parameter 1^ℓ, produces a run which is not in γ, i.e., the goal γ is not achieved in that run, but the run is nevertheless accepted by J.

Definition 2 (Computational verifiability [10]). Let P be a protocol with the set of agents Σ. Let $\delta \in [0,1]$, $J \in \Sigma$, $\psi \in \mathscr{F}_{\mathsf{hon}}$, and γ be a property of P. Then, we say that *the goal γ is guaranteed in P by ψ and δ-verifiable by J* if for every instance π of P the following conditions are satisfied:

(i) If $\pi \models \psi$, then $\Pr[\pi(1^\ell) \mapsto (J : \mathsf{accept})]$ is overwhelming as a function of ℓ.

(ii) $\Pr[\pi(1^\ell) \mapsto \neg\gamma,\ (J : \mathsf{accept})]$ is δ-bounded as a function of ℓ.

Just as in the case of accountability, requiring the probability in (ii) to be negligible, i.e., $\delta = 0$, is too strong for many reasonable protocols. It is shown in [10] that verifiability is a special case of accountability (see also our analysis of Helios below).

VI. Analysis of the Original Variant of Helios

In this section, we formally analyze accountability and verifiability of the original version of the Helios voting system with homomorphic aggregation of ballots, as described in Section III; the variant with mixnets can be analyzed similarly and such an analysis should yield similar results.

A. The Protocol Model

Following Section III-A, the system can be described in a straightforward way as a protocol in the sense of Section V-A. However, we fix some more details.

The participants in this system are the following:

– The election administrator A, who is assumed to be honest, i.e., in the terminology of Section V-A, Π_A is required to contain only the honest program of A.
– The election server S with a bulletin board BB, which can be honest or dishonest, i.e., Π_S contains, besides the honest programs of S and BB, all probabilistic polynomial-time programs (with the appropriate interface to the rest of the system); we assume, however, that once the content of the bulletin board is published, the bulletin board presents the same content to every party and, moreover, every party is able to obtain this content. In what follows, we do not distinguish the election server S and the bulletin board BB and use these terms interchangeably.
– The (human) voters $v_1,\dots,v_n$, which, similarly to S, can be honest or dishonest.
– Browsers (client programs) of voters $b_1,\dots,b_n$, which can be honest or dishonest.
– External verifying programs of voters $vp_1,\dots,vp_n$, which are assumed to be honest (this is a reasonable assumption, as they may be provided by independent parties).
– The authentication authority AA, which can be honest or dishonest.
– Decryption trustees $D_1,\dots,D_m$, which can be honest or dishonest.

The election administrator first creates an election by posting certain parameters, such as a list of candidates (choices or questions) and the list of eligible voters. Then, the decryption trustees create in a distributed manner a public/private key pair for a homomorphic encryption scheme, such that every decryption trustee holds a share of the private key. The public key is sent to the administrator, who publishes a signed fingerprint of the election, i.e., a signed hash over all parameters, including the public key.

The authentication authority is supposed to send to every voter a credential (password), which she needs to submit when casting a ballot.

In the voting phase, a voter can abstain from voting or decide to vote (for some candidate). We model this choice by a probability distribution $p = p_0,\dots,p_l$, where l is the number of candidates, such that a voter abstains from voting with probability p_0 and votes for candidate i, for $i \in \{1,\dots,l\}$, with probability p_i.

A voter who has decided to vote, prepares her ballots using her browser. A ballot consists of the encrypted choice of the voter and a zero-knowledge proof that the ballot is well-formed.

As already described in Section III-A, voters can choose to either cast a ballot or audit it. We parameterize the system by a sequence of probabilities $\vec{q} = q_1, q_2, \dots$, where q_i is the probability that an honest voter, when she gets to decide whether to audit her i-th ballot or not, chooses audit. The level of accountability/verifiability Helios provides will depend in such a sequence.

If a voter eventually decides to cast a ballot, this ballot is submitted to the authentication authority who authenticates the voter (requiring the password) and then is supposed to forward this ballot to the election server. A voter who wants to vote but did not get a credential from the authentication authority would complain.

We note that the voter behavior we model here is slightly simplified to a real voter. For instance, a voter who decides not to vote (this decision is taken with probability p_0) also does not prepare or audit any ballots. However, a real voter could prepare and audit some ballots, but then decide not to actually cast a ballot. Also, as already mentioned, we do not allow voters to revote. We emphasize that while these simplifications ease the presentation of the model and the analysis, they do not change our negative or positive results.

After the voting phase is finished, the content of the bulletin board (containing voter name/ballot pairs) is published and signed by the election server. Voters can now check this content. We model this step by parameterizing the system with a probability q_{check} that an honest voter checks the bulletin board: A voter who has voted checks that her ballot appears on the bulletin. A voter who has not voted makes sure that no ballot with her name appears on the bulletin board, but that she obtained a credential from the authentication authority that would have enabled her to vote (and complains otherwise). Note that due to that procedure, ballot stuffing is not possible (more precisely, highly risky): As the list of eligible voters is correctly published, the adversary cannot add a vote for a non-eligible voter. Further, if the adversary votes using a name of an abstaining voter, this is detected with probability q_{check}.

Finally, the published ballots are then multiplied and the resulting ciphertext is decrypted by the decryption trustees.

We denote the protocol modeled as described above by $\mathsf{P}_{\mathsf{Helios}}(n, \vec{q}, q_{check}, p)$ where n is the number of voters, $\vec{q}$, q_{check} and p are as described above. So far we have not specified a verification or judging procedure, i.e. the steps that a protocol participant or an observer can take to determine whether a run should be accepted or rather some verdict should be stated. Such a procedure will be specified below.

We take the standard cryptographic assumptions for the used cryptographic primitives, in particular, digital signatures, zero-knowledge proofs, and homomorphic encryption. However, our proofs for accountability/verifiability do not require encryption to ensure confidentiality of the plaintext. (Clearly, this would be needed for vote privacy.)

B. Properties of the Protocol

Goal. Ideally, one might expect the system to provide individual accountability (see Section V-B) whenever the goal γ_{opt} is violated, where γ_{opt} contains all runs in which the result returned in the tallying phase exactly corresponds to the input of all the voters. However, as already noted in [10], this goal is too strong for almost all real voting systems, and it is in fact too strong for Helios, as it is impossible to give any guarantees concerning dishonest voters: a dishonest voter might indicate to the (dishonest) bulletin board that she is not going to check her receipt and that the bulletin board is free to change her ballot.

Therefore, the best goal γ we can hope for is that the result is correct up to the votes of dishonest voters. More formally, γ is satisfied in a run if the published result exactly reflects the actual votes of the honest voters in this run and votes of dishonest voters are distributed in some way on the candidates, possibly in a different way than how the dishonest voters actually voted. We analyze Helios with respect to this goal. If this goal is achieved, one can be sure that all votes of honest voters are counted correctly and that votes of dishonest voters are counted at most once.

For the analysis of voting systems it is instructive to also consider a family of goals γ_k, where γ_k coincides with γ except that up to k of the votes of *honest* voters (rather than only dishonest voters) may be altered as well; obviously $\gamma = \gamma_0$. Note that since honest voters check their receipts only with a certain probability, undetected altering of votes by voting authorities/machines may occur. (We will exactly measure the probability for such things to happen in our analysis.)

Problems. In some situations, it is impossible to determine who misbehaved. This problem will be reflected in the relatively weak accountability property we can state for Helios, which is far from corresponding to individual accountability. This also influences the level of verifiability one can state for Helios. Intuitively, the main problems are as follows:

Problem 1. If a voter v_i complains that the bulletin board does not show her ballot (correctly) or that a ballot is shown even though the voter did not vote, it is unclear who cheated: it may be the voter (who possibly has falsely claimed that something went wrong), the browser of the voter (which possibly has not submitted any ballot or has submitted a wrong ballot), the bulletin board (which has possibly changed/discarded the ballot), or the authentication authority (which might have used the voters password to vote). Hence, a judge can in this case only state $\mathsf{dis}(v_i) \vee \mathsf{dis}(b_i) \vee \mathsf{dis}(BB) \vee \mathsf{dis}(AA)$. Such a (necessarily) coarse verdict has serious negative consequences: It is impossible to punish a particular party. Hence, the involved parties are not deterred from misbehaving. Moreover, it is not clear what to do after such a verdict is stated. If we abort the election process in such a case, then any dishonest voter can easily obstruct the whole election. If we ignore voters' complaints and just continue the process, then we lose verifiability.

Problem 2. If a voter complains that the auditing of a ballot failed, it is not clear whether the browser actually manipulated the ballot or whether the voter complained for no reason. Therefore, the judge can only state $\mathsf{dis}(v_i) \vee \mathsf{dis}(b_i)$. However, this problem is less severe than the previous one, since a voter could switch to another browser (client program) and/or to another machine and try again.

Problem 3. If a voter complains that she did not get a credential from the authentication authority, it is not clear whether the authority in fact did not send a credential or whether the voter complained for no reason. Therefore, the judge can only state $\mathsf{dis}(v_i) \vee \mathsf{dis}(AA)$. However, this problem is, again, less severe than the first one, since a voter could try to somehow obtain a credential before the voting phase ends.

Judging Procedure. In order to be able to formally state and prove the level of accountability Helios provides, we need to define a judging procedure which decides whether to accept a run or whether to blame (groups of) parties. Such a procedure should, in fact, be part of the protocol specification.

The judging procedure is based solely on publicly available information, and hence, can be carried out both by an external judge and a regular protocol participant. The procedure consists of the following steps, where we assume that the procedure is run honestly by some party a.

J1. If a participant b deviates from the protocol in an obvious way, e.g., a decryption trustee refuses to contribute to the decryption or a zero-knowledge proof is not correct, then a states $\mathsf{dis}(b)$.

J2. If a voter v_i complains that her ballot does not appear (correctly) on the bulletin board or a ballot appears even though v_i did not vote, a states the verdict $\mathsf{dis}(v_i) \vee \mathsf{dis}(b_i) \vee \mathsf{dis}(BB) \vee \mathsf{dis}(AA)$, as explained above. We denote the set of runs in which some voter complains in that way by α_{compl}.

J3. If a voter v_i complains that the auditing of a ballot failed, a states $\mathsf{dis}(v_i) \vee \mathsf{dis}(b_i)$, as explained above. We denote this event by $\alpha_{v-compl}$.

J4. If a voter v_i complains that she did not get a credential from the authentication authority, a states $\mathsf{dis}(v_i) \vee \mathsf{dis}(AA)$, as explained above. We denote this event by α_{cred}.

J5. If none of the above happens, a accepts the run.

Let a be a regular protocol participant or an observer. By $\mathsf{P}^a_{\mathsf{Helios}}(n,\vec{q},q_{rec},p)$ we denote the protocol $\mathsf{P}_{\mathsf{Helios}}(n,\vec{q},q_{rec},p)$, where the agent a carries out the judging procedure described above, possibly in addition to the program a runs anyway. We assume a to be honest, in particular, to run the judging procedure as described.

Accountability. Due to the problems described above, with the first problem being the most severe one, we can only prove accountability of the considered variant of Helios for a relatively weak accountability property Φ_k (that reflects these problems). More precisely, Φ_k consists of the following accountability constraints:

$$\begin{aligned}
\alpha_{compl} &\Rightarrow \mathsf{dis}(v_1) \vee \mathsf{dis}(b_1) \vee \mathsf{dis}(BB) \vee \mathsf{dis}(AA) \mid \dots \\
&\quad \cdots \mid \mathsf{dis}(v_n) \vee \mathsf{dis}(b_n) \vee \mathsf{dis}(BB) \vee \mathsf{dis}(AA) \\
\alpha_{v-compl} &\Rightarrow \mathsf{dis}(v_1) \vee \mathsf{dis}(b_1) \mid \dots \\
&\quad \cdots \mid \mathsf{dis}(v_n) \vee \mathsf{dis}(b_n) \\
\alpha_{cred} &\Rightarrow \mathsf{dis}(v_1) \vee \mathsf{dis}(AA) \mid \dots \\
&\quad \cdots \mid \mathsf{dis}(v_n) \vee \mathsf{dis}(AA)
\end{aligned}$$

$$\begin{array}{c} \neg\gamma_k \cap \neg\alpha_{compl} \cap \\ \neg\alpha_{v\text{-}compl} \cap \neg\alpha_{cred} \end{array} \Rightarrow \begin{array}{l} \mathsf{dis}(BB) \mid \mathsf{dis}(AA) \\ \quad \mid \mathsf{dis}(D_1) \mid \cdots \mid \mathsf{dis}(D_m). \end{array}$$

We now define the parameter δ^k for the system $\mathsf{P}^a_{\mathsf{Helios}}(n,\vec{q},q_{rec},\vec{p})$ and the goal γ_k, which captures the probability that something went wrong (γ_k was not achieved), but nobody was blamed. (Recall that k is the tolerated number of incorrectly counted votes of honest voters.) We will later consider a special case of the formula for δ^k, which is easier to grasp. However, let us note already that δ^k decreases exponentially in k.

$$\delta^k = \sup_{j_1,\dots,j_n \in \mathbb{N}} \left(\left(\sum_{m=1}^{n} q^*_{j_m} \prod_{i=m+1}^{n} \tilde{q}^*_{j_i} \right) + (1-q_{check}) \prod_{i=1}^{n} \tilde{q}^*_{j_i} \right)^{k+1}$$

where $q^*_i = q_1 \cdots q_{i-1} \cdot (1-q_i)$ is the probability that an honest voter casts her i-th ballot, and hence, audits the previous $i-1$ ballots, and $\tilde{q}^*_i = \sum_{j<i} q^*_j$ is the probability that an honest voter casts the j-th ballot, for some $j < i$. For the case $i = 0$, we set $q^*_i = 0$ and $\tilde{q}^*_i = 1$.

Now we are ready to formally state the level of accountability for the considered variant of Helios, where a proof sketch of the theorem is given in the appendix.

Theorem 1. *Let a be an external judge or a voter, and let $k \le n$. The agent a ensures (Φ_k, δ^k)-accountability for $\mathsf{P}^a_{\mathsf{Helios}}(n,\vec{q},q_{rec},\vec{p})$.*

This theorem implies that, in $\mathsf{P}^a_{\mathsf{Helios}}$, the probability that the goal γ_k is not achieved but a nevertheless does not blame anybody is at most δ^k, up to some negligible value. Moreover, if the goal is not achieved ($\neg\gamma_k$), then a single agent can be held accountable (and because of fairness rightly so) if no voter complained that her ballot does not appear on the bulletin board ($\neg\alpha_{compl}$), no voter complained that auditing of a ballot failed ($\alpha_{v\text{-}compl}$), and no voter complained that she did not get a credential ($\neg\alpha_{cred}$); as explained, in the other cases, blaming individual parties is not possible.

As mentioned, since the expression δ_k, as defined above, may be difficult to grasp, we consider a special case for which this expression becomes easy to understand. In this special case, we assume that a voter, in each round, audits a ballot with the same probability $q \in [0,1]$, i.e., we assume that $q_i = q$ for all i. With this, we can show:

Corollary 1. *Let $q \in [0,1]$ and let $\vec{q} = q_1, q_2, \dots$ with $q_i = q$ for all i. Let a be an external judge or a voter, and let $k \le n$. The agent a ensures (Φ_k, δ^k_q)-accountability for $\mathsf{P}^a_{\mathsf{Helios}}(n,\vec{q},q_{check},\vec{p})$, where*

$$\delta^k_q = \max(1-q_{check}, 1-q)^{k+1}.$$

Note that this formula, which decreases exponentially in k, reflects the following intuition: If, in each round, the voter audits her ballot with the same probability, and if the adversary tries to change a vote, he either changes/adds a ballot on the bulletin board (which goes undetected with probability $1 - q_{check}$) or he lets the browser encrypt a different vote. In the latter case, as the probability for getting detected is the same in each round, the adversary can simply change the first ballot (which goes undetected with probability $1-q$); waiting for other later ballots does not increase the adversaries chances of not being caught.

Verifiability. Let us first observe that since a ensures (Φ_k, δ^k)-accountability, a also ensures $(\{\neg\gamma_k \Rightarrow \psi\}, \delta^k)$-accountability, where $\psi = \bigvee_{b \in \Sigma} \mathsf{dis}(b)$ and Σ is the set of all participants in $\mathsf{P}^a_{\mathsf{Helios}}(n, \vec{q}, q_{rec}, \vec{p})$, because whenever a states some verdict ψ', then ψ' implies ψ. Let us also observe that the judging procedure (that a runs) is constructed in such a way that a accepts a run if and only if a does not blame anybody. Altogether, this means that if the goal γ_k is not achieved, a will blame someone (i.e., not accept the run), except with probability at most δ^k (plus some negligible function). Conversely, if all parties are honest, a will accept the run (because of the fairness condition). But this means, we obtain the following level of verifiability of $\mathsf{P}^a_{\mathsf{Helios}}(n, \vec{q}, q_{rec}, \vec{p})$.[5]

Corollary 2. *Let a be an external judge or a voter. In $\mathsf{P}^a_{\mathsf{Helios}}(n, \vec{q}, q_{check}, \vec{p})$, the goal γ_k is guaranteed by $\bigwedge_{a \in \Sigma} \mathsf{hon}(a)$ and δ^k-verifiable by a.*

This corollary reflects the weakness of the system $\mathsf{P}^a_{\mathsf{Helios}}(n, \vec{q}, q_{rec}, \vec{p})$ already discussed for accountability: by wrongly complaining, every single dishonest voter can spoil the election process, i.e., a cannot accept the run. Conversely, only if all parties are honest, including all voters, it is guaranteed that a can accept the run. So, the level of verifiability provided by the original variant of Helios is very good in terms of the probability that cheating is detected, but quite weak in terms of the guarantees that an election observer, a, will accept a run. (Ideally, in a voting system, it should suffice if only the voting authorities are honest, in order to ensure that the election is successful, i.e., is accepted by observers.).

VII. Accountability and Verifiability of Helios with Detached Names

In this section, we analyze accountability of the variant of Helios with detached names, with and without the countermeasure described in Section III. However, we consider only the variant where the names of the voters who voted are published on the bulletin board (although detached from the ballots). In the variant where these names are not published at all, ballot stuffing cannot be prevented, and hence, this version would not provide a reasonable label accountability and verifiability. We start with a description of the protocol model.

A. The Protocol Model

The protocol model is very similar to the original variant, as described in Section VI-A, except for the following differences:

(a) The names of voters are not published on the bulletin board linked to their ballots, they are rather published independently.
(b) In the version with the additional voter-provided randomness, the voters provide their randomness in the voting phase, as described in Section III-C.

We denote the variant of the protocol without voter provided randomness by $\mathsf{P}^a_{\mathsf{Helios-DN}}(n, \vec{q}, q_{check}, p)$ and the one with voter provided randomness by $\mathsf{P}^a_{\mathsf{Helios-DNVR}}(n, \vec{q}, q_{check}, p, r)$, where $\vec{q}$, q_{check}, and $\vec{p}$ are defined as in Section VI and r is a probability distribution on the set of valid inputs (e.g. alphanumeric strings), determining the way honest voters choose their random input.

B. Negative Result

While it was already argued in Section III-B that the variant without voter-provided randomness does not ensure a sufficient level of verifiability, we now make this statement precise. We show that even the very weak accountability property Φ^*_k consisting of the constraint

$$\neg\gamma_k \Rightarrow \bigvee_{b \in \Sigma} \mathsf{dis}(b),$$

where Σ contains every participant of the protocol is not ensured. This accountability constraint only requires that somebody is blamed if γ_k is not fulfilled, and thus, it exactly corresponds to what verifiability requires (as discussed at the end of Section VI-B). More precisely, we show that this constraint can be violated with probability close to 1.

Recall the clash attack for the variant of Helios with detached names from Section III-B. The result of this attack was that all the voters who vote for the same candidate and have audited their ballots the same number of times cast identical ballots and obtain identical receipts. If c_0 denotes a fixed candidate, the bulletin board can therefore safely replace, for each group of duplicates for candidates different than c_0, all members of this group, except for one member, and turn these members into ballots for c_0. We let $\mathsf{safe}(k)$ denote the event that when the adversary follows this strategy he manages to change at least k votes of honest voters.

Let $P[\mathsf{safe}(k)]$ denote the probability that $\mathsf{safe}(k)$ holds true. Now we immediately obtain:

Theorem 2. *Let a be an external judge or a voter. The agent a does not ensure (Φ^*_k, δ)-accountability for $\mathsf{P}^a_{\mathsf{Helios-DN}}(n, \vec{q}, q_{check}, \vec{p})$ for any $\delta < P[\mathsf{safe}(k+1)]$.*

For typical cases, the probability $P[\mathsf{safe}(k)]$ is close to 1, if only k is not too close to the number of honest voters. Hence, the variant of Helios with detached names (and without the countermeasure applied) provides virtually no accountability/verifiability. For instance, in an election with three candidates, 100 voters, with a uniform distribution on the candidates (including the choice of abstaining from voting), and the probability $\frac{1}{2}$ that a voter audits a ballot (i.e., $\frac{1}{2} = q_1 = q_2 = \cdots$), the probability $P[\mathsf{safe}(20)]$ that the adversary can change as much as 20 votes without a blaming anybody is about 0.96.

C. Positive Result

We now show that $\mathsf{P}^a_{\mathsf{Helios-DNVR}}(n, \vec{q}, q_{check}, p, r)$ provides a similar level of accountability as the original variant of Helios (see Section VI). More precisely, the accountability property is exactly the same, namely Φ_k, and, for realistic distributions r of voter-provided input, the δ-value is only slightly worse than

[5]A general theorem showing that accountability implies verifiability was proved in [10], Proposition 1.

δ^k; but clearly δ now heavily depends on r. We denote the δ-value for Helios with detached names and the countermeasure applied by δ^k_{DNUR} (see below for the definition).

To define δ^k_{DNUR}, let j denote some fixed candidate (intuitively, the favorite candidate of the adversary). We now consider the following scenario. We assume that all n voters, the authentication authority, and the trustees are honest, and that all browsers and the bulletin board are dishonest. The browsers, controlled by the adversary, always use the same randomness for computing all ballots. The intuition is that by this the adversary produces the maximum number of duplicate ballots. (However, he always takes the voter-provided randomness into account as prescribed.) Now, the bulletin board, which is also controlled by the adversary, replaces, for each group of duplicates for candidates different than j, all members of this group, except for one member, and turns these members into ballots for j. In other words, we consider a scenario where the adversary tries to generate as many duplicates as possible and turns as many of these duplicates as possible into ballots for j without being detected.

Now, by R^j_l we denote the probability that in the above scenario, the adversary can turn *exactly* l ballots for candidates different from j into ballots for j. It is easy to see that R^j_l only depends on the probability distributions r and p.

Now, we define (see below for an explanation):

$$\delta^k_{DNVR} = \max_j \sum_{l=0}^{n} R^j_l \cdot \delta^{k-l},$$

where j ranges over all candidates, δ^i for $i \geq 0$ is defined as in Section VI, and $\delta^i = 1$ for $i < 0$.

With Φ_k being the accountability property defined in Section VI, we can show:

Theorem 3. *Let a be an external judge or a voter. The agent a ensures* $(\Phi_k, \delta^k_{DNVR})$*-accountability for* $\mathsf{P}^a_{\mathsf{Helios-DNVR}}(n, \vec{q}, q_{check}, \vec{p}, r)$.

The intuition behind the definition of δ^k_{DNVR} is as follows: The best strategy of an adversary to change $k+1$ votes into votes for j is to first change as many votes as possible in a safe way by turning duplicates into votes for j, as described above. If he can change l votes in this way, then still $k+1-l$ more votes have to be changed in order to altogether change at least $k+1$ votes, and hence, violate the goal γ_k. For these remaining $k+1-l$ votes the chances of being able to change these votes without being detected is δ^{k-l} as in the original variant of Helios. For the first l votes the probability is R^j_l. Note that since the list of voters that actually voted is published, ballot stuffing is prevented for similar reasons as in the case of the original variant of Helios.

Clearly, δ^k_{DNVR} highly depends on the entropy of the randomness provided by the voters: if they always choose the same "random" value, $\mathsf{P}^a_{\mathsf{Helios-DNVR}}(n, \vec{q}, q_{check}, \vec{p}, r)$ is essentially the same as $\mathsf{P}^a_{\mathsf{Helios-DN}}(n, \vec{q}, q_{check}, \vec{p})$ (and hence does not provide any reasonable level of accountability/verifiability).

However, for a realistic entropy of the voters input, the level of accountability provided by the considered system, as given by Theorem 3, is quite close to the level of accountability provided by the original variant of Helios. For instance, let us consider an election with two candidates, 500 voters, a uniform distribution on the candidates (including abstention of voting), and $q_{check} = q_1 = q_2 = \cdots = \frac{1}{2}$. Then, if the voters uniformly draw their random input from a dictionary of size 100000 (or using some other distribution providing with the same entropy, which seems to be a realistic setting), then δ^k_{DNVR} is 0.565, 0.286, 0.036, and 0.0011 for k equal to 1, 2, 5, and 10, respectively. The level of accountability provided by the original variant, as given by Corollary 1, is, for the same parameters, 0.5, 0.25, 0.03125, and 0.00097; hence, quite similar. In both cases, the probability that more than 10 votes can be changed is very close to zero.

Verifiability. Analogously to Section VI-B, we obtain the following corollary for the verifiability of the system.

Corollary 3. *Let a be an external judge or a voter. In* $\mathsf{P}^a_{\mathsf{Helios-DNVR}}(n, \vec{q}, q_{check}, p, r)$*, the goal* γ_k *is guaranteed by* $\bigwedge_{a \in \Sigma} \mathsf{hon}(a)$ *and* δ^k_{DNVR}*-verifiable by a.*

VIII. Accountability and Verifiability of Helios with Aliases

In this section, we analyze the variant of Helios with aliases, as described in Section III, with respect to accountability and verifiability. We show that the clash attack is indeed reflected in a very low level of accountability/verifiability formally expressed by a very high δ, which is almost 1. We also show that the fix proposed in Section III provides a very effective countermeasure. We start with a description of the protocol model.

A. The Protocol Model

The protocol model is very similar to the original variant, as described in Section VI-A, with the following differences:

- After the list of eligible voters is published, but before the registration phase begins, *AA* publishes the list of aliases (the number of aliases should be the same as the number of eligible voters).
- In the registration phase, the voters do not only get a password, but also an alias from the authentication authority, which needs to be provided along with the password for authentication when a ballot is cast.
- If a voter who wants to vote has not received any alias, she complains. Similarly, a voter who decides not to vote, checks, with probability q_{check}, whether she has obtained her alias and whether this alias appears on the bulletin board. Note that this procedure prevents ballot stuffing: The list of aliases the adversary publishes cannot be longer than the list of eligible voters. Also, the adversary cannot add votes for aliases he did not publish before, and it would be risky to vote for aliases that abstained from voting, as these abstaining voters also check the bulletin board (with probability q_{check}).

Besides the problems concerning accountability already mentioned in Section VI-B, an additional problem occurs in the alias variant: if the same alias appears twice on the bulletin board, then it is not clear whether the bulletin board misbehaved or whether the authentication authority has assigned this alias to two different voters. However, this problem is not very severe, since both parties are part of the election authority.

We take care of this problem by the following changes in the judging procedure:

J5. If the same alias appears twice on the bulletin board, then a states $\mathsf{dis}(BB) \vee \mathsf{dis}(AA)$. We denote the event that the same alias appears twice on the bulletin board by α_{twice}.

J6. If none of the above happens, a accepts the run.

We denote this variant of Helios by $\mathsf{P}^a_{\mathsf{Helios-AL}}(n,\vec{q},q_{check},p)$ and $\mathsf{P}^a_{\mathsf{Helios-ALF}}(n,\vec{q},q_{check},p)$, respectively, depending on whether the unfixed or fixed version is considered, where in the fixed version, as described in Section III-C, voters check the freshness of ballots.

As discussed in Section III-B, we assume that the adversary knows, for some honest voters, how those voters are going to vote. We will represent this knowledge as sets $G_1,\dots,G_l$ of voters, where l is the number of candidates and G_i contains those voters that are going to vote for candidate i.

B. Negative Result

We now show that for any $\delta < 1$ the version of Helios with aliases does not ensure even the very weak accountability constraint Φ^*_k introduced in Section VII-B, which only requires that somebody is blamed if the goal is violated.

In our clash attack on Helios with aliases (Section III-B) the adversary issues the same alias to all the members of the same group G_i (as defined above). Let K denotes the maximal number of votes that can be safely changed using the knowledge represented by the sets $G_1,\dots,G_l$. For instance, if $G_1 = \{\}$, $G_2 = \{a,b,c\}$ and $G_3 = \{d,e\}$, then $K = 3$, since, e.g., the adversary could change the votes of b, c, and e. Now, the following theorem is easy to see. It also immediately carries over to the verifiability of the system.

Theorem 4. *Let a be an external judge or a voter. Let $k+1 \leq K$. The agent a does not ensure (Φ^*_k,δ)-accountability for $\mathsf{P}^a_{\mathsf{Helios-AL}}(n,\vec{q},q_{check},p)$ for any $\delta < 1$.*

C. Positive Result

We now show that the level of accountability of $\mathsf{P}^a_{\mathsf{Helios-ALF}}(n,\vec{q},q_{check},p)$ is comparable to the one for the other two (fixed) variants of Helios: the accountability property is almost the same and the δ-value is also very good, although the concrete values differ.

As the general formula for δ for the considered variant of Helios is even more complex than the formula for δ^k for the original variant, for brevity of presentation, in this section, we focus only on the special case where $q_i = q$ for all i, i.e., honest voters always audit ballots with probability q; see our technical report [11] for the general case. Let $G_1,\dots,G_l$ and K be defined as above. Further, let the accountability property Φ'_k consist of the constraints for α_{compl}, $\alpha_{v-compl}$, and α_{cred} as defined in Section VI-B and in addition the constraints:

$$\alpha_{twice} \Rightarrow \mathsf{dis}(BB) \vee \mathsf{dis}(AA)$$

$$\begin{array}{c} \neg\gamma_k \cap \neg\alpha_{compl} \cap \neg\alpha_{twice} \cap \\ \cap \neg\alpha_{v\text{-}compl} \cap \neg\alpha_{cred} \end{array} \Rightarrow \begin{array}{r} \mathsf{dis}(BB) \mid \mathsf{dis}(AA) \mid \\ \mathsf{dis}(D_1) \mid \cdots \mid \mathsf{dis}(D_m). \end{array}$$

Theorem 5. *Let a be an external judge or a voter. The agent a ensures (Φ'_k,δ^k_{ALF})-accountability for $\mathsf{P}^a_{\mathsf{Helios-ALF}}(n,\vec{q},q_{check},p)$, where*

$$\delta^k_{ALF} = (1 - q \cdot q_{check})^{k+1}.$$

Note that δ^k_{ALF} decreases exponentially in k. The proof of this theorem can be found in our technical report [11].

Similarly to the previous sections, we obtain, as a corollary, that in $\mathsf{P}^a_{\mathsf{Helios-ALF}}(n,\vec{q},q_{check},p)$, the goal γ_k is δ^k_{ALF}-verifiable and guaranteed by the set of all protocol participants.

IX. Conclusion

In this paper, we showed that a simple attack, which we call a clash attack, can successfully be mounted on four different e-voting systems (and variants thereof), some of which have been deployed in real elections. This illustrates that the clash attack is a potentially dangerous attack for a large class of e-voting systems which try to provide verifiability by letting voters check receipts on a bulletin board. It is interesting future work to see to which other systems the clash attack could be applied. We hope that our findings raise the awareness of this attack and that future e-voting systems provide mechanisms to prevent this attack, or if trust assumptions are made which prevent the attack (e.g., aliases are distributed honestly), that the scope of these assumptions is made clear.

For the four systems we studied, we proposed countermeasures for the clash attack, where some of these countermeasures only require changing the audit procedure clerks and/or voters have to carry out, others change the voting procedure itself.

For the different variants of Helios (with the countermeasures applied, if necessary)—the original variant, the variant with aliases, and the variant with detached names—we provided fine-grained analysis results, which highlighted some further problems with the accountability of Helios orthogonal to the clash attack and which precisely captured the different levels of accountability and verifiability the different variants of Helios and the proposed countermeasures provide.

This formal analysis is of independent interest since such an analysis has rarely been carried out for concrete e-voting systems, even more so in a cryptographic model which allows precise measurement of the level of accountability/verifiability a system provides. Also, while verifiability is a very prominent property of e-voting systems, accountability has obtained much less attention, both in informal and formal treatments.

References

[1] Ben Adida. Helios: Web-based Open-Audit Voting. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[2] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jaques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *USENIX/ACCURATE Electronic Voting Technology (EVT 2009)*, 2009.

[3] Josh Benaloh and Eric Lazarus. The Trash Attack: An Attack on Verifiable Voting Systems and a Simple Mitigation. Technical Report MSR-TR-2011-115, Microsoft, 2011.

[4] J.H. Fund. *Stealing elections: how voter fraud threatens our democracy*. Encounter Books, 2004.

[5] IACR 2011 Election, 2011. http://www.iacr.org/elections/2011/candidates.html.

[6] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant Electronic Elections. In *Proceedings of Workshop on Privacy in the Eletronic Society (WPES 2005)*, pages 61–70. ACM Press, 2005.

[7] Simon Kramer and Peter Y. A. Ryan. A modular multi-modal specification of real-timed, end-to-end voter-verifiable voting systems. In *2011 International Workshop on Requirements Engineering for Electronic Voting Systems (REVOTE 2011)*, pages 9–21. IEEE Computer Society, 2011.

[8] Steve Kremer, Mark Ryan, and Ben Smyth. Election Verifiability in Electronic Voting Protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *15th European Symposium on Research in Computer Security (ESORICS2010)*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[9] R. Küsters, T. Truderung, and A. Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *IEEE Symposium on Security and Privacy (S&P 2011)*. IEEE Computer Society, 2011.

[10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 526–535. ACM, 2010.

[11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. Technical Report 2012/116, Cryptology ePrint Archive, 2012. http://eprint.iacr.org/2012/116/.

[12] R. L. Rivest and W. D. Smith. Three Voting Protocols: ThreeBallot, VAV and Twin. In *USENIX/ACCURATE Electronic Voting Technology (EVT 2007)*, 2007.

[13] Ron Rivest. The ThreeBallot Voting System. http://people.csail.mit.edu/rivest/Rivest-TheThreeBallotVotingSystem.pdf, October 1, 2006.

[14] Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards Automatic Analysis of Election Verifiability Properties. In Alessandro Armando and Gavin Lowe, editors, *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, Lecture Notes in Computer Science. Springer, 2010.

[15] Mehdi Talbi, Benjamin Morin, Valérie Viet Triem Tong, Adel Bouhoula, and Mohamed Mejri. Specification of electronic voting protocol properties using adm logic: Foo case study. In Liqun Chen, Mark Dermot Ryan, and Guilin Wang, editors, *Proceedings of the 10th International Conference Information and Communications Security (ICICS 2008)*, volume 5308 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2008.

[16] Wombat Voting system, 2011. http://www.wombat-voting.com/.

Appendix A
Proof Sketch of Theorem 1

Due to the space limitations, we will only provide a sketch of the proof for $k = 0$ to give some intuition about how δ^k is obtained.

Let $S = \mathsf{P}^a_{\mathsf{Helios}}(n, \vec{q}, q_{rec}, \vec{p})$. Since it is quite easy to show that a is computationally fair in S, we only focus on the completeness condition of Definition 1, which requires that for all of the accountability constraints in Φ_0 the probability that a does not ensure this constraint is at most δ^0 plus some negligible function.

Clearly, by the definition of the judging procedure, a always ensures the first, the second and the third accountability constraint. Therefore, we focus on the probability that the fourth constraint is not ensured, which is the case if (i) the goal γ_0 is violated, i.e. the adversary changed at least one (honest) vote, (ii) no voter complains that her ballot does not appear on the bulletin board ($\neg\alpha_{compl}$), that the auditing of a ballot failed ($\alpha_{v\text{-}compl}$), or that she did not get a credential ($\neg\alpha_{cred}$), and (iii) no individual party is blamed. We show that this happens only with probability at most δ^0 (plus some negligible value).

First, let us notice that, if the adversary wants to change a vote, then every time an honest voter, say the i-th voter, initiates her voting procedure, the adversary faces the following choice: He can either (A) carry on the correct voting procedure without changing the voters choice or he can (B) decide to *try* to change the j_i-th ballot of the voter. Therefore, without loss of generality, we can represent a strategy of the adversary as a sequence of integers $\vec{j} = j_1, j_2, \ldots$ representing the choice of the adversary: $j_i = 0$ meaning that the adversary carries out the correct voting procedure for the i-th voter and $j_i > 0$ meaning that the adversary will try to change her j_i-th ballot.

Now, if the adversary runs (B) for the i-th voter, then one of the following may happen:

(a) The voter submits her j-th ballot, by which the adversary changes the vote and achieves his goal (he does not need to change further votes); this happens with probability q^*_j.
(b) The voter submits one of the first $j_i - 1$ ballots and the adversary does not get the chance to change the j-th ballot. This case happens with probability $\tilde{q}^*_j$.
(c) The voter audits her j-th ballot and the attack is detected (the goal is not achieved).

If, after all the voters have voted, the goal is still not achieved, the adversary has to replace a ballot on the bulletin board (otherwise he could be used to forge zero-knowledge proofs), which is detected with probability $1 - q_{rec}$.

We obtain the following probability that the adversary succeeds using strategy $\vec{j}$:

$$\sum_{m=1}^{n} \left(q^*_{j_m} \cdot \prod_{i=1}^{m-1} \tilde{q}^*_{j_i} \right) + (1 - q_{check}) \cdot \prod_{i=1}^{n} \tilde{q}^*_{j_i}.$$

For a given m, the subformula $q^*_{j_m} \cdot \prod_{i=1}^{m-1} \tilde{q}^*_{j_i}$ expresses the probability that this strategy succeeded, because case (a) happened for the m-th voter, and case (b) happened for all previous voters. The subformula $(1 - q_{check}) \cdot \prod_{i=1}^{n} \tilde{q}^*_{j_i}$ expresses the probability that the strategy succeeded because (b) has happened for all voters and then the adversary successfully replaced one ballot on the bulletin board.

From the above formula, we obtain δ^0 by taking the supremum over all possible strategies $\vec{j}$, as described above.