A simple alternative to Benaloh challenge for the cast-as-intended property in Helios/Belenios

Véronique Cortier, Jannik Dreier, Pierrick Gaudry, Mathieu Turuani

A simple alternative to Benaloh challenge for the cast-as-intended property in Helios/Belenios

Véronique Cortier, Jannik Dreier, Pierrick Gaudry, Mathieu Turuani

Université de Lorraine, CNRS, INRIA, LORIA

November 5, 2019

Abstract

Helios is a well-established online voting protocol which has been used for different real world elections, and which aims at ensuring – among other properties – verifiability. Belenios is a variant of Helios that brings eligibility verifiability and a threshold decryption à la Pedersen to distribute trust.

In both protocols, the cast-as-intended property, part of verifiability, is obtained using a cast-or-audit technique known as Benaloh challenge: each voter can choose to either audit her encrypted ballot, or to submit it. However, an audited ballot can never be submitted, as the audit breaks vote secrecy. This means that in particular none of the ballots which are actually submitted and counted have been audited.

We propose a simple variant, that might be better suited in some situations. In our solution, each vote is partially audited before being submitted. The partial audit ensures that errors are detected with a certain probability, without breaking secrecy.

1 Introduction

Helios is a well established online voting protocol [1, 2] which has been used for different real world elections [2, 3], and which aims at ensuring – among other properties – verifiability and vote secrecy.

Verifiability is meant to ensure the correctness of the result independently of any software or hardware: the voting protocol provides audit procedures and cryptographic proofs allowing to verify the correctness of the result. Verifiability is often split in different sub-properties according to the auditors that perform the checks: *individual verifiability* (each voter can check that her ballot was tallied), *universal verifiability* (anyone can check that the result corresponds to all published ballots), and *eligibility verifiability* (anyone can check that ballots come only from eligible voters). Sometimes verifiability is also split into three different notions corresponding to the different phases of the election, in total ensuring *end-to-end verifiability*: *cast-as-intended* (ensuring that the encrypted ballot contains the vote intended by the voter), *recorded-as-cast* (the vote recorded by the voting protocol corresponds to the one cast by the voter), and *counted-as-recorded* (all recorded votes have been correctly tallied).

Belenios [4] is a variant of Helios that adds eligibility verifiability and a threshold decryption à la Pedersen to distribute trust.

In both protocols, the cast-as-intended property is obtained using a cast-or-audit technique known as Benaloh challenge: each voter can chose to either audit her encrypted ballot, or to submit it (cut-or-choose approach). In case of an audit, the protocol has to provide a proof that the encrypted ballot contains a correct vote for the candidate chosen by the voter. In Helios and Belenios, this is done by revealing the randomness used in the encryption. Using these values, one can recompute the encryption, and check whether the same ciphertext is obtained. This recomputation of the encryption using the randomness must, in principle, be

done by a party or a tool that is independent of the voting server and of the voting device. An approach is to use a third-party web service hosted by an independent entity, that offers to perform these computations.

However, an audited ballot can never be submitted, as the audit breaks vote secrecy. Hence, if a voter chooses to audit his vote, he needs to "vote again", i.e., prepare a new ballot. This means that in particular none of the ballots which are actually submitted and counted have been audited. However, as the protocol never knows in advance whether a certain ballot will be audited or not, it cannot cheat (i.e., encrypt a different vote) without risking being caught in an audit.

Although Benaloh challenge is in principle a good way to guarantee the cast-as-intended property, the question of its practical usability is often raised (see for example the recent study [6]). In the reference implementation of Belenios it was in fact not included.

We propose a simple variant which might be better suited in some situations. In our solution, each vote is partially audited before being submitted. The partial audit ensures that errors are detected with a certain probability, but without breaking secrecy.

Assuming that there are k candidates in the election, the general idea is that the voting device will not only encrypt the vote v but also a random value s and the masked vote $m = v - s \mod k$, yielding two additional ciphertexts β and γ together with a zero-knowledge proof that guarantees the relation $v = m + s \mod k$. The voter will (privately) see the values v, s, and $v - s$ and will audit at random either β or γ, checking that β (resp. γ) indeed corresponds to s (resp. $v - s$). This audit does not reveal any information on v (this is a form of one-time pad) and can therefore be delegated to anyone. Thus the server simply displays the audited value (the plaintext and the corresponding randomness) and anyone can check that this is correct, while the voter has just to check that he sees the clear value s (resp. $v - s$).

This approach has the following advantages: all votes are necessarily audited, whereas in the classical Benaloh challenge the audit is optional. In particular in Helios and Belenios, voters that do not understand the importance of the audit might decide to directly submit their votes, without any audit. On the contrary, in our solution, all ballots are necessarily audited. This also streamlines the voting process, as there are less choices for the voter. In this sense the new process might be easier to explain to voters, although it might be more difficult to understand why the audit is convincing, as it is only partial.

2 An always-audit-and-cast approach

2.1 Description of the protocol

We concentrate on the part of Helios that we modify, namely the sub-protocol where the voter cast her ballot and checks that everything went correctly for her. The setup of the election is the same, and therefore a public encryption key `pk` is known and must be used to encrypt the ballots. The encryption algorithm is ElGamal, so that we can have homomorphic properties and easy zero-knowledge proofs. The post-process once the election is closed is also the same, and the tally can be done with two main techniques: either homomorphic decryption, where the ballots are agglomerated before the result being decrypted; either individual ballot decryption after shuffling using verifiable mixnets. As usual, we assume that there is a public bulletin board `BB` that is append-only and to which only the server can write.

The protocol is summarized in Figure 1.

Ballot preparation and pre-casting. We assume for the moment that the choice v of a voter is an integer between 0 and $k - 1$ (there are k choices), and this is the input that the voter gives to her voting device (in addition to her authentication data). In order to build her ballot b, the voting device will encrypt v and also s and $(v - s) \bmod k$, where s (called the shift) is a random integer in $[0, k - 1]$. This yields three ciphertexts α, β, γ. The knowledge of s alone obviously does not leak any information on v, nor does unique knowledge

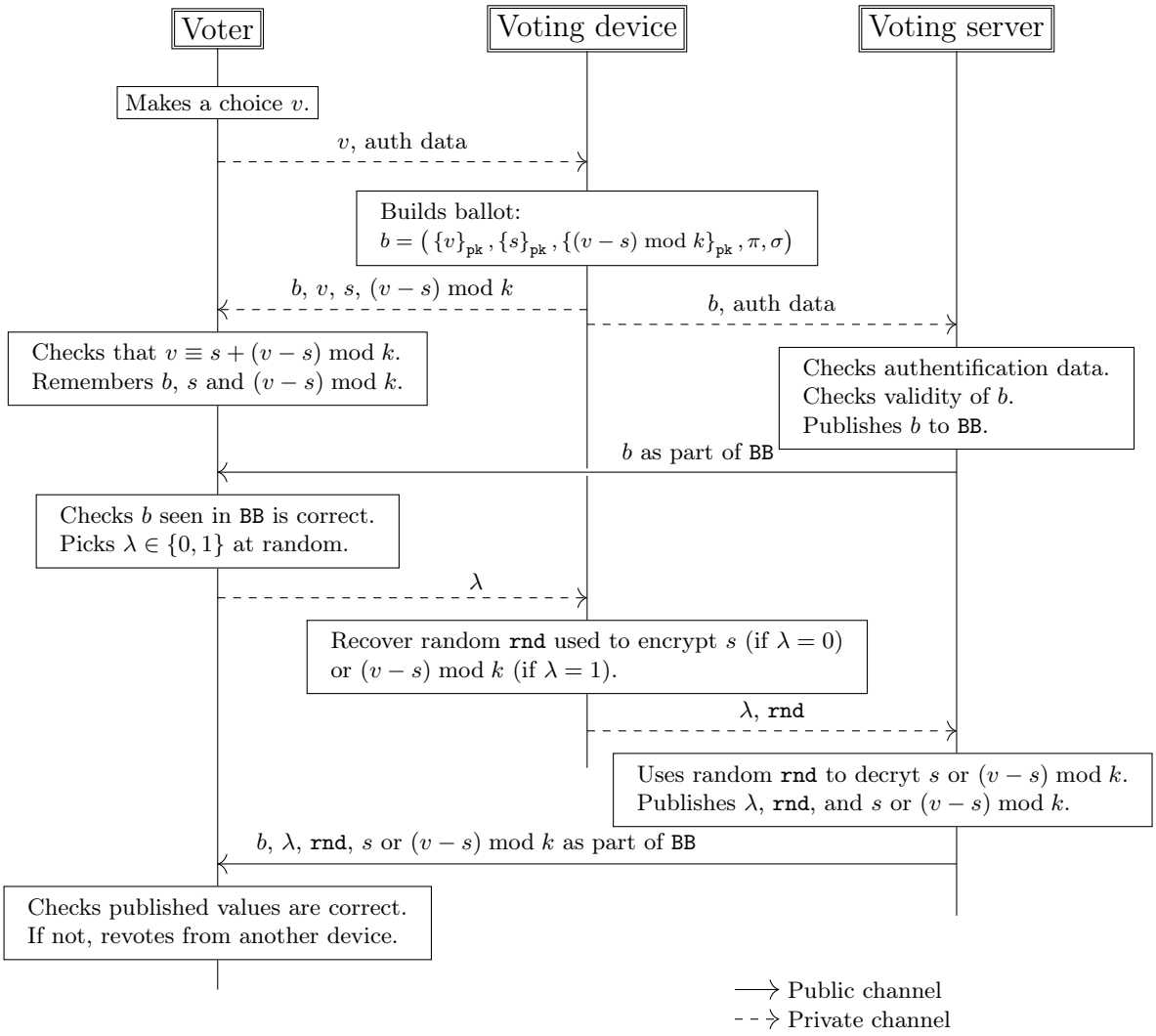

Figure 1: The audit-and-cast part of the protocol

of $(v-s) \bmod k$ that is a one-time-padded encryption of v. The voting device will then create a zero-knowledge proof π that the encryption α of v is well-formed, and that the sum of the two cleartexts corresponding to β and γ indeed corresponds (modulo k) to the value of the cleartext contained in α. In the case of Belenios, it will also compute a signature σ of all this with the credential provided by the voter. In summary, a ballot has the form

$$b = \left(\{v\}_{\mathtt{pk}}, \{s\}_{\mathtt{pk}}, \{(v-s) \bmod k\}_{\mathtt{pk}}, \pi, \sigma\right),$$

where v is the vote, s is a random shift, k is the number of choices, π is a zero-knowledge proof and σ is a signature.

The voting device shows to the voter the three values v, s and $(v-s) \mod k$, together with the hash of the ballot b that is going to be sent to the voting server. At this point, this is important that the voter keeps the hash of the ballot for future check of its presence in the ballot box, but also that she checks that indeed s and $(v-s) \mod k$ sum up to v modulo k and she must remember s and $(v-s) \mod k$ (in this order).

In the meantime, the voting device has sent the ballot b to the server that checks its validity and publishes it to the public board BB.

Voter's audit of the ballot. The voter must then proceed to auditing her ballot. First, as in Helios and Belenios, she checks that the hash of her ballot is indeed present on the public board BB. This check must be done independently of the voting device. Ideally, the voter will visit a webpage using a different physical device that the one used for voting (maybe using her smartphone, while she uses a browser on a personal computer as a voting device), but visiting the webpage using a static link, not provided by the voting interface of the voting device can already prevent several attack scenarios on the personal computer.

Thereafter (and only thereafter), the voter chooses a bit $\lambda \in \{0, 1\}$ at random and sends it to the voting device. The voting device forwards this value of λ to the voting server together with the value of the random rnd that has been used to encrypt s (if $\lambda = 0$) or $(v - s) \mod k$ (if $\lambda = 1$), thus revealing one (and only one) of the last two encrypted values of the voter's ballot.

When receiving this information, the voting server uses the random value rnd to decrypt the corresponding data (no key is needed), and publishes λ, rnd and the cleartext s or $(v-s) \mod k$ on BB, as a side-data of the already published ballot.

The voter then looks at the board BB and checks that aside her ballot there is the same value of λ than the one she gave to her voting device, and that the published value s or $(v-s) \mod k$ is the one she remembers from the ballot preparation phase. If everything is fine, then this is the end, otherwise, she must alert the organizers about a potential problem with this type of device and re-vote from another voting device.

General audit of the public ballot box. Like in Helios and Belenios, the security relies on a general audit of the ballot box. Additionally to the usual validity checks, it is good if as many auditors as possible check for each ballot, not only all the zero-knowledge proofs and the signature, but also that the published values of λ and of rnd lead to the published value of s of $(v - s) \mod k$ when used to decrypt the appropriate part of the published ballot b. We did not include this check as part of the voter's task, since everyone can do it, but again, as in Helios and Benelios, any voter is welcome to perform a general audit of the whole process of the election.

End of the election – decryption. Once all the validity checks have been done, only the $\{v\}_{\mathtt{pk}}$ parts of the ballots are kept for tallying. The decryption keys must not be used to decrypt the other parts of the ballots. And of course, the $\{v\}_{\mathtt{pk}}$ themselves must not be decrypted. Instead, tally is performed thanks to either an homomorphic operation to agglomerate the encrypted votes or a shuffle based on verifiable mixnets. Both techniques provide a proof of correct decryption.

2.2 Security claims

We spell out the security properties *intended* by our protocol. This is not supported by any proof nor any deep analysis. Therefore these properties should be taken with care and can only be used at your own risk.

First of all, we think that this variant inherits the standard properties of Helios/Belenios, namely vote secrecy and individual and universal verifiability (as well as eligibility verifiability for Belenios).

- *vote privacy:* No one can learn any information on how someone voted, even in presence of a corrupted voting server and corrupted voters. Note that technically, Helios does not satisfy vote privacy due to an attack [7] that introduces a bias in the result. Belenios does not suffer from this issue thanks to the signature mechanism.

- *individual verifiability:* Provided that a voter checks that her ballot b is on the bulletin board, she is guaranteed that it will be tallied.

- *universal verifiability:* the result of the election corresponds to the content of the ballots of the board. This is guaranteed thank to the proofs of correct decryption.
- *eligibility verifiability:* ballots only come from legitimate voters. In Belenios, this is guaranteed thanks to the signature of the ballots.

Additionally, our variant guarantees *cast-as-intended*: any voter that audits her ballot is guaranteed that her ballot encodes her intended vote, even if her voting device and the voting server are corrupted and collude.

To explain the reason why this is the case, let us consider an attacker that controls the voting device and the voting server, and that wants to cast a valid ballot containing a vote different from the one intended by the voter. Let b be a valid ballot, voting for v, and let b' be a fake ballot, voting for a value v' different from v.

During the interaction between the voting device and the voter (see Figure 1), the attacker must decide to display either b or b'. If b is displayed, since later on, the voter is going to check that it is indeed in the public board `BB`, there won't be any possibility to change its value. So the attacker has to display b' in order to be able to cheat. In this first message, the voting device also displays the values v, s and $(v-s) \mod k$. Here, the attacker can not display v' in place of v (this is the very first thing that the voter will check), but he could display different values for s and $(v-s) \mod k$. The best interest of the attacker is to display the values that are encrypted in the ballot b', namely s' and $(v'-s') \mod k$. Note that not both of them can correspond to the ones in b, because they should sum up to v modulo k, which is different from v'. Therefore the attacker will construct the ballot b' with the same value of $s' = s$ or the same value $(v'-s') \mod k = (v-s) \mod k$.

Then the attacker must make a choice about what to display during the first exchange with the voter. Let us study the two possible scenarios:

- If the voting device displays b', v, s and $(v-s) \mod k$, then this passes the arithmetic check made by the voter. However, during the audit phase, if the λ value picked by the voter corresponds to the place where the pairs $[s, (v-s) \mod k]$ and $[s', (v'-s') \mod k]$ differ, this will be revealed: the voter will see that the value published on `BB` is different from the one she had remembered. Assuming that the voter picks λ uniformly in $\{0,1\}$, the fraud will be detected with probability $1/2$.
- If the voting device displays b', v, s' and $(v'-s') \mod k$, then the arithmetic check will fail. The attacker will succeed only if the voter did not perform it, or was not confident enough with her calculation to detect the problem. In that case, the success probability of the attacker is hard to predict, because it highly depends on whether or not the voter does a careful arithmetic check.

From this discussion, we see that a fraud will be detected with a high probability. Assuming the arithmetic check is easy enough to perform for the typical voter of the election, then the attacker will be detected with a $1/2$ probability at each attempt.

Gaining a significant advantage by changing N ballots will raise on average $N/2$ alerts from the voters for which the audit phase failed. The probability of raising no alerts decreases exponentially with N.

2.3 The zero-knowledge proof

Mixnet-based decryption. When the decryption is done individually, there is no restriction on the format of the vote v encrypted in the ballot. It is then rather natural to encode v as an integer, and for our protocol to be realistic, we need to assume that this integer is bounded by a reasonably small integer k, because the voter will need to perform arithmetic with integers of that size, preferably as a mental calculation or with a pocket calculator (or a basic smartphone application emulating the same). A good example is when there are k candidates and the voter must choose her favorite. Therefore v will just be a number between 0 and $k-1$ encoding the choice.

In this case, the zero-knowledge proof π will be comprised of

- a proof that v belongs to $[0, k-1]$;
- after an homomorphic computation of the encryption of $s + ((v-s) \bmod k) - v$, prove that this is the encryption of either 0 or k.

These proofs are simple and classical when the encryption is done with ElGamal. We remark that since v and s are in $[0, k-1]$, the "$\bmod k$" operation that reduces $(v-s)$ in the same interval can either be a no-op, or an addition of k, thus explaining the two possible choices in the second proof.

Homomorphic tally. In some cases the result of the election can be obtained as a simple linear function of the ballots v. The basic example is a yes/no question where the result is the number of votes for each choice. It is also the case for slightly more complicated elections where the voter can choose t out of n candidates, maybe with some minimum and maximum values for t. The general method for this is to decompose the choice into several bits v_0, v_1, $\ldots$, v_{n-1}, where v_i denotes whether the candidate number i is selected, so that to get the score of the candidate i, we can homomorphically compute the encryption of the sum of all the v_i in all the ballots, and then decrypt only this sum.

The basic adaptation of our protocol to this setting is to replicate n times the auditing made by the voter. However, this would require her to perform n additions modulo 2. We propose as an alternative the following. Let $v = \sum v_i 2^i$, which is an integer less than 2^n, whose encryption can be computed homomorphically from the encryptions of the v_i. Then the ballot can take the form

$$b = \left(\{v_0\}_{\mathtt{pk}}, \{v_1\}_{\mathtt{pk}}, \ldots, \{v_{n-1}\}_{\mathtt{pk}}, \{s\}_{\mathtt{pk}}, \{(v-s) \bmod 2^n\}_{\mathtt{pk}}, \pi, \sigma \right),$$

where the shift s is a random integer in $[0, 2^n - 1]$. In that case, the proof π must include all the conditions that have to be satisfied by the v_i's, and of course a proof that s and $(v-s) \bmod 2^n$ indeed sum up to the value of v modulo 2^n. With this strategy, the additional cost of our protocol in terms of size of the ballot is moderate: there are two additional ElGamal encrypted blocks, and a simple additional zero-knowledge proof.

However, in general the voter's audit of the ballot becomes more difficult. Checking that s and $(v-s) \bmod 2^n$ sum up to v modulo 2^n is maybe feasible, especially if we replace 2^n by the smallest power of 10 larger than 2^n. However, the conversion between the choices of the voter, namely a subset of the candidates, and the value of the integer v encoding this subset is not accessible to a layman. If the voter must choose one and exactly one candidate, then this becomes simple, though.

3 Discussion

3.1 Advantages and disadvantages

We list some of the good and bad properties of our variant, especially in comparison to an approach based on Benaloh challenge.

Advantages:

- All the voters are supposed to perform the audit. Even if some of them will probably not perform the arithmetic check, there is no way for the server or the device to guess whether the check is indeed done. Furthermore, the real ballot is audited, not a mock ballot as in a Benaloh challenge.
- The system is robust to a moderate bias in the randomness when picking the bit λ.
- If all entities are honest, the voter's experience is completely linear, with 4 sequential steps, while for Benaloh challenge, the voter's experience is (in theory) an unbounded loop of sequential steps.

- There is no need to add a third-party to the protocol.
- There is no need to add new cryptographic tools compared to what is already present in Helios and Benelios.
- The risk of auditing the real ballot and leaking the voter's choice, present with Benaloh challenge if the tool is not clear enough, disappears.

Drawbacks:

- The protocol requires the voter to perform a modular addition. By taking the modulus to be a power of 10, this might be acceptable, but still needs some explanation.
- This hardly convinces layman voters that this indeed brings the cast-as-intended property. Maybe it is easier to get a security feeling with Benaloh challenge.
- This is only mildly compatible with homomorphic tallying.
- This does not bring privacy w.r.t the voting device (Benaloh challenge does not either).

3.2 On the necessity of including $\{v\}_{\mathtt{pk}}$ in the ballot

In the case where the tallying is done by individual decryption after shuffling with verifiable mixnets, it is tempting to have simpler ballots of the form

$$b = \left(\{s\}_{\mathtt{pk}}, \{(v-s) \bmod k\}_{\mathtt{pk}}, \sigma\right),$$

that do not contain the encryption of the vote $\{v\}_{\mathtt{pk}}$, and do not include any zero-knowledge proof.

Indeed, the encrypted choice $\{v\}_{\mathtt{pk}}$ or $\{v+k\}_{\mathtt{pk}}$ can be computed homomorphically from the other two encrypted parts. And this is the data that is going to be shuffled and then decrypted. After decryption, the $+k$ can easily be removed if necessary. A zero-knowledge proof for the relation between the vote and the two committed values is no longer required because it is computed in a way which guarantees it.

Of course, this would allow a voter (or her voting device) to produce invalid ballot that are detected as such only late in the process, after the election is closed.

But more critically, privacy is not preserved in this setting. Let us assume that there are two choices, hence $k = 2$, and two voters, Alice and Bob. Alice's choice is $v_A = 0$ and she chose $s_A = 1$, so that $(v_A - s_A) \bmod 2$ is 1 in her case. Bob's choice is $v_B = 1$ and he chose $s_B = 0$, so that $(v_B - s_B) \bmod 2$ is 1 in his case. We further assume that both Alice and Bob pick $\lambda = 0$ during the audit phase, so that $s_A = 1$ and $s_B = 0$ are publicly revealed. The votes that are computed homomorphically are $v'_A = 2$ and $v'_B = 1$. After shuffling and decrypting, anyone will see that the (unordered) multi-set of encrypted votes is $\{1, 2\}$, that must be interpreted modulo 2. An attacker can then deduce that the person who voted 2 (hence 0) can not be Bob, since $s_B = 0$ can not produce an underflow during the subtraction. Hence he learns that Alice voted for 0 and Bob voted for 1.

Adding the $\{v\}_{\mathtt{pk}}$ in the ballot allows to avoid leaking any information about the possible underflow, thus fitting in the one-time-pad setting.

References

[1] Ben Adida, Olivier de Marneffe, and Olivier Pereira. Helios voting system. `http://www.http://heliosvoting.org`.

[2] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association.

[3] International association for cryptologic research. Elections page at `https://www.iacr.org/elections/`.

[4] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for Helios under weaker trust assumptions. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS'14)*, volume 8713 of *LNCS*, pages 327–344, Wroclaw, Poland, September 2014. Springer.

[5] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 409–423, Dallas, USA, 2017. ACM.

[6] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. What did I really vote for? On the usability of verifiable e-voting schemes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI'18, pages 176:1–176:13. ACM, 2018.

[7] Peter Roenne. Private communication. The attack has been discovered by Peter Roenne and then described in the paper [5].