Known caveats of Belenios 3.1

Véronique Cortier and Pierrick Gaudry, CNRS, Loria

April 28, 2025

Abstract

We list here some known caveats of Belenios, as implemented in v3.1 (and earlier versions). We concentrate on issues that do not appear in the theoretical description of the (lastest version of) Belenios; typically, we do not recall that Belenios does not provide coercion-resistance beyond allowing revotes, nor cast-as-intended.

We do not know whether these caveats will be fixed, nor how, since the development of the Belenios software is now independent of our research teams.

We assume here that the reader is already familiar with Belenios, whose high level description can be found in [4].

1 Verifiability issue when revoting

As discovered in [2], there is a verifiability attack when the voting server is compromised or, to some extent, when the attacker fully controls the communications. The attack works roughly as follows:

- Alice votes for candidate A and submits a ballot b_A.
- Alice then changes her mind and votes for candidate B and submits a ballot b_B.
- Alice checks that b_B appears on the bulletin board.
- The voting server replaces b_B by b_A. This is interpreted as a revote, hence the vote of Alice for candidate B is replaced by a vote for A.

Discussion. If we assume that:

- Alice checks that her ballot appears on the bulletin board each time she votes, hence in particular when she submits ballot b_A,
- and if the ballot box is constantly monitored,

then the monitoring would catch the replacement of b_B by b_A and declare the election invalid. Hence there is no attack under these 2 assumptions.

Now, if Alice only checks her last vote and if the voting server correctly guesses that Alice will not check her first vote, then this attack cannot be detected in Belenios 3.1 and earlier. Note that this attack applies to Helios [1] as well.

As initially suggested in [2], a way to fix this attack is to link a ballot with the previous ones by including a hash of the previous ballots inside the zero-knowledge proof of the new ballot. The fix proposed in [2] assumes that voters vote sequentially, which is not realistic. Hence [3] (briefly) devises a refined fix for Belenios, where ballots are linked with previous ballots *of the same voter*.

2 No proper Bulletin Board

Belenios strongly assumes that all participants (voters, decryption trustees, registrar, auditors) have access to a public, append-only bulletin board. In practice, the implementation of the bulletin board is

simply a webpage served by the voting server. Hence a dishonest voting server may provide inconsistent views to the participants. The consequences in terms of security are discussed in [5].

How to securely realize a public bulletin board typically assumes a distributed setting, with several online servers, as discussed in [5]. An alternative would be that the voting server *signs* a chained list of the ballots, providing some *accountability*: if the voting server misbehaves, it would be eventually caught. Then it could be possible to achieve security against a *malicious but cautious* voting server that would not take the risk of signing inconsistent bulletin boards. The details of such an approach remain to be worked out.

3 Fragile vote privacy

Vote privacy is ensured in Belenios by distributing the secret decryption key amongst decryption trustees, possibly in a threshold manner. Each trustee generated their private key share on their own machine. Hence the complete decryption key is never present on a single computer. Of course, the trustees should use their decryption key only to decrypt the final bulletin board, otherwise vote privacy may be compromised.

With the Belenios voting platform, all the operations (key generation and decryption) can be done by the trustees in their browser. They should simply store their secret key in a secure manner. This means however that the protocol is not strictly implemented as explained in the literature.

Homomorphic tally. When the questions of an election are of the form: "select between k and n candidates in a list", Belenios uses El Gamal encryption in an homomorphic manner: during the tally, ballots are combined together (by multiplying them) so that only the final result needs to be decrypted. From a theoretical perspective, election trustees should look at the bulletin board, check that each individual ballot is valid, compute the homomorphic combination of the ballots, and decrypt it (and provide a proof of correct decryption).

With the Belenios voting platform, each trustee does not check any individual ballot. They are provided with the (claimed) homomorphic combination C of the ballots, by the server, who may behave dishonestly. Hence a dishonest server may try to provide another ciphertext instead, for example, Alice's ballot.

Therefore, trustees should first check that the combination C is the one that appears on the main page of the election (in the gray area). Then assuming in addition that the election page is monitored, then the other checks can indeed be discharged to the monitoring and privacy is still preserved.

Mixnet tally. When voters need to rank or grade the candidates of an election, Belenios uses a mixnet-based tally: ballots are shuffled and re-randomized, with a proof of correct shuffling, in a consecutive manner, by each of the decryption trustee (the administrator may chose to skip some trustee). Then the resulting ballots are decrypted one by one by the trustees, with a proof of correct decryption.

From a theoretical perspective, each trustee t should:

- retrieve the content of the ballot box on the public bulletin board,
- check the validity of each ballot
- check the proofs of shuffles of the previous shuffles
- produce a shuffle S_t and provide a proof of correct shuffling
- once the shuffling phase is over, retrieve the list C of ciphertexts to be decrypted
- check that it corresponds to the chain of shuffles from the initial ballot box
- decrypt it and provide a proof of correct decryption

For usability reasons, what is really done by each trustee t is:

- retrieve the content of the ballot box on the public bulletin board

- produce a shuffle S_t (+ proof) and store a hash of it
- once the shuffling phase is over, retrieve the list C of ciphertexts to be decrypted and store a hash of it
- decrypt it + provide a proof of correct decryption

The other checks are discharged to the auditors. If trustees properly follow the instructions described in `https://www.belenios.org/instructions.html`, privacy should not be broken. Indeed, before decrypting, a trustee should check on the web page of the election that:

- S_t appears on the web page (to make sure their shuffle has not been ignored, otherwise privacy could be completely broken)
- C appears on the web page (to make sure they will decrypt the correct list, otherwise they may be decrypting the initial list of ballots).

Note that these checks are not explained by the web client of the trustees so they should be properly instructed. Alternatively, on the long term, it would make sense to provide trustees with a native application (instead of a webpage) that would perform all the required checks, without having to assume an external, active, auditor.

References

[1] Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX*, 2008.

[2] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Election Verifiability Revisited: Automated Security Proofs and Attacks on Helios and Belenios. In *34th IEEE Computer Security Foundations Symposium, (CSF 2021)*, pages 1–15, 2021.

[3] Véronique Cortier, Alexandre Debant, and Vincent Cheval. Election verifiability with proverif. In *36th IEEE Computer Security Foundations Symposium (CSF'23)*, 2023.

[4] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, pages 214–238. Springer, 2019.

[5] Lucca Hirschi, Lara Schmid, and David A. Basin. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *Computer Security Foundations Symposium (CSF)*, 2021.