Improved Verifiability for BeleniosVS

Thomas Haines[1] and Rajeev Goré[2]*

[1] Australian National University, Canberra, Australia
[2] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract. The BeleniosVS electronic voting scheme offers an attractive mix of verifiability and privacy properties. Moreover, using the ProVerif protocol-verification tool, BeleniosVS has automatic machine-aided analysis of (end-to-end) verifiability in 96 different threat models with the machine-aided analysis finding proofs in 22 cases and finding attacks in the remaining 74 cases. The high number of threat models covered by ProVerif delivers a much richer security analysis than the norm.
We revisit the BeleniosVS scheme and propose several refinements to the ProVerif security model and scheme which increase the number of threat models in which the scheme has verifiability from 22 to 28. Our new ProVerif security model also implies end-to-end verifiability but the requirements are easier to satisfy. Interestingly, in all six improvements, both the changes to the security model and one or more changes to the scheme are necessary to prove verifiability.

Keywords: Verifiability · Machine-checked proofs · ProVerif · BeleniosVS

1 Introduction

Secure electronic voting is a difficult security problem with numerous competing constraints. The key approach to securing electronic elections is so-called "end-to-end verifiable electronic voting" which is usually broken down into three sub-properties: namely, cast-as-intended, collected-as-cast and counted-as-collected. Cast-as-intended captures the idea that the encrypted ballot the voter casts contains the vote she intended. Collected-as-cast captures the idea that the cast ballot was collected by the election authority without tampering. Counted-as-collected captures the idea that the collected ballots are properly counted. Our work focuses on the end-to-end verifiability of the BeleniosVS voting scheme [6], which is the culmination of a line of schemes starting with Helios [1].

BeleniosVS, like all voting schemes, should provide verifiability and privacy under reasonable assumptions. Privacy captures the intuition that nothing more should be leaked about the voter's ballot than can be discerned from the outcome of the election; for the rest of this paper we will eschew further discussion

* Rajeev Goré acknowledges the support of the National Centre for Research and Development NCBR, Poland, under the PolLux/FNR-CORE project STV (POLLUX-VII/1/2019).

of privacy. BeleniosVS has end-to-end verifiability under certain trust assumptions; that is, under certain trust assumptions we can be assured that for every honest voter, the encrypted ballot in the tally which belongs to her (because the accompanying signature verifies with her verification key) does indeed decrypt to the ballot she believes she cast. As we have previously said, BeleniosVS has been analysed in 96 different threat models, 22 of which have verifiability which makes providing a simple description of the assumptions problematic.

Informally, BeleniosVS achieves end-to-end verifiability under certain assumptions as follows:

1. it achieves cast-as-intended because a voter can check that the cast ballot contains the vote she intended under the assumption that either the auditing device or registrar—which we will formally introduce in section 2.1—is honest;
2. it achieves collected-as-cast since the voter can check that the ballot posted by the voting server was the one she cast, under the assumption that the registrar is honest; and finally
3. it achieves tallied-as-collected because a voter can check the zero-knowledge proofs produced by the talliers which ensure the published ballot is correctly decrypted and tallied.
4. moreover, the authentication measures (signatures) prevent the ballot box from being stuffed unless the registrar is corrupt.

Any claimed verifiability and privacy properties must assume some underlying security model. There are two prominent such models called the symbolic (Dolev-Yao [11])) model and the computational model (more specifically, most voting schemes are analysed in the Random Oracle Model (ROM) [3]). The symbolic model assumes that the underlying cryptographic primitives are black-boxes (perfectly secure), and the attacker can only use these primitives as such. On the other hand, the computational model assumes that all data types are bit-strings and all computation is done by probabilistic polynomial-time Turing machines. The computational model captures more attacks but is harder to reason about; both models miss certain classes of attacks (such as timing attacks).

The authors of Belenios provided mathematical proofs of the correctness of the claimed verifibility and privacy of Belenios using ProVerif [5], an automatic theorem prover for verifiying properties of security protocols which uses the symbolic model. In contrast, tools such as Coq [4] and EasyCrypt [2] work in the computational model. We also use ProVerif for our work.

ProVerif is sound wrt the symbolic model: that is, if it finds an attack or concludes there isn't one then we can be sure that these conclusions are true in the symbolic model. However, one challenge in using ProVerif for complicated schemes is its incompleteness: that is, there are queries on which ProVerif may not terminate, and hence be unable to decide whether or not there is an attack, in which case, we learn nothing by using ProVerif. Due to this, BeleniosVS [6] was not able to be directly proved end-to-end verifiable in ProVerif:

rather the authors introduced sufficient conditions for verifiability; these conditions are checked by ProVerif. Both the sufficient conditions and the definition to which they refer have the advantage over prior definitions of enforcing vote eligibility; that is the definitions prevent the stuffing of the ballot box with unauthorised votes. However, this comes at a high cost because the definition no longer distinguishes between systems where attacks are detected and system where attacks are not detected but rather deems both to be insecure (Sec. 3). We present new related conditions which do distinguish between detected and undetected attacks but at the cost of not capturing ballot stuffing; any scheme proved secure under our new conditions should separately be proved to be free of ballot stuffing. It remains an open problem to devise a definition which combines the strengths of both the original definition in [6] and ours.

In general, theorems about verifiability (in ProVerif) are either about the unreachability of certain "undesirable" states or about correspondence assertions; we will be interested in correspondence assertions. These are of the form "if event e has occurred then event e' has occurred previously". For example, a state where the voter n believes she voted for a particular candidate c should imply that the ballot published next to her name is for candidate c. To encode this into ProVerif, let VERIF(n,c) denote that voter n believes she voted for candidate c and let TALLY(n, c) denote that the ballot published for voter n decrypts to candidate c. Then we model our previous claim by saying that VERIF(n,c) $\Rightarrow$ TALLY(n, c); that is if a voter believes a certain vote is cast then it was. ProVerif attempts to prove that this correspondence holds for all executions of the protocol or alternatively to find an execution which does not satisfy the correspondence. (For those interested in the technicalities we recommend reading the ProVerif manual [5]).

ProVerif identifies many attacks on BeleniosVS in the ProVerif security model, but some of these attacks are not "real' because they cannot manifest in the implementation of BeleniosVS; the false "attacks" identified by ProVerif would have been detected by the voter during the voting phase. For example, the adversary votes on behalf of the voter (without detection) but does not fake the voting server's acknowledgment when the voter tries to vote (the lack of which the voter will detect). The six threat models in which we improve on the previous analysis are all interesting since they require both our changes to the security model and to the scheme. Specifically in all six, ProVerif correctly identified that the scheme was insecure (both in the ProVerif model and with respect to end-to-end verifiability); however, the attacks ProVerif found would have been detected by the voter and hence are not proper attacks on verifiability. If we introduce our improved security model ProVerif still finds attacks (albeit more complicated than the previous attacks). The more complicated attacks were the inspiration for our improvements to the scheme which ultimately increase the security.

1.1 Contribution

- We identified that the definition of end-to-end verifiability in [6] does not meaningfully capture verification failures; this results in a definition which is much closer to integrity than verifiability. The use of the definition to analysis verifiability returns false attacks which obscures the real attacks and hinders the analysis of improvements to the scheme.
- We refined BeleniosVS to include a defence-in-depth approach, using various overlapping independent means to mitigate attacks. Even if the voter chooses not to verify the voting sheet, the voting device and server (if honest) will still perform certain checks and notify the voter whether or not they (the voting device and server) believe the ballot was cast successfully. Consequently, in our refined scheme, verifiability may hold even if the adversary can cast ballots on behalf of the voter, since the attack will still be detected (depending on the exact threat model).
- The ProVerif encoding of cast-as-intended in [6] required the property to hold even if the verification checks failed. We changed the property so that no guarantees are provided if verification fails. This weakens the security requirement while preserving the sufficiency for verifiability (though not for the definition of verifiability in [6])
- We refined the channel analysis to be more consistent. Our refined analysis highlights the importance of keeping basic protections like TLS and digital certificates.

Our more nuanced sufficient conditions ensure that the attacks found by ProVerif work against the real scheme and are not artefacts of our model and verifiability definition. These attacks highlighted edge case vulnerabilities which we prevent by introducing simple mechanisms. Overall, our refinements to the security model and scheme increase the number of threat models in which the scheme has verifiability from 22 to 28.

2 Background

In Helios [1], the election authority sends an email to each eligible voter containing instructions on how to construct a valid ballot. The voter constructs her ballot on her personal computing device, which encrypts the ballot using the public key of a special voting server run by the election authority, it also constructs (zero-knowledge) proofs that the encrypted ballot is well-formed wrt the instructions. She sends her ciphertext to the voting server over the internet, thus the voting server collects the (encrypted) ballots and proofs from all voters. These encrypted ballots and proofs are published by the voting server, and the published ballots are decrypted and tallied by a group of (physically distributed) authorities called the talliers. While the voting server has some means (normally email address) to authenticate votes from eligible voters, there is no external mechanism (such as signatures) to authenticate that the ballots cast actually came from eligible voters.

The Belenios family of voting schemes originated with Belenios [10] which is a more secure version of Helios [1] with better authentication of ballots. Specifically in Belenios, each voter has a signing key—and verification key—which the voters use to sign their encrypted ballot and proofs. Anyone can now check that the submitted ciphertext came from an eligible voter by verifying the signature with the voter's verification key. However, a coercer could simply demand that the voter divulge the randomness used in the encryption of their vote, enabling the coercer to learn how that voter voted once the ballots are published.

The next version, BeleniosRF [7], replaced the signatures and proofs with rerandomisable alternatives, which allow the voting server to randomise the ciphertexts, proofs, and signatures before publishing. Now, the coercer cannot be certain that the randomness is what the voter claims it to be, so the voter cannot be coerced to prove how she voted using this tactic. BeleniosVS [6] extends BeleniosRF by including a cast as intended mechanism in which the voter can check that the cast ciphertext decrypts to the vote she intended to cast. However, due to the rerandomisation, she cannot check the ballot posted by the voting server was the one she cast. Being unable to perform a direct collected-as-cast check in either scheme complicates the end-to-end verifiability claims of these schemes.

2.1 BeleniosVS

For completeness we will introduce the BeleniosVS protocol here; the original description can be found in [6]. We will avoid giving in full all the technical details that are not relevant to our contributions.

The protocol involves seven distinct primary entities.

The election administrator publishes the parameters of the election including the list of eligible voters and candidates.

The registrar generates the voter credentials and voting sheets and issues them to the voters. A voter's voting sheet contains encryptions—of the candidates—which are signed using her credentials.

The voter uses her voting device to cast a ballot from the voting sheet she received from the registrar with the password she received from the voting server. Optionally she can use her auditing device to verify the correctness of the voting sheet.

The voting device receives the password and ballot from the voter which it submits to the voting server.

The auditing devices check the correctness of the voting sheet.

The voting server performs some basic checks and if they pass, posts the ballot to the bulletin board.

The bulletin board is public and can be audited by anyone.

The tallying authorites takes the ballots from the bulletin board from which they compute and publish the result.

The system proceeds in the following three phases.

Setup phase: The election administrator publishes the election parameters. Then the tallying authorities generate the encryption public key and secret keys. The registrar then prepares the voting sheets by generating signing credentials (keys) for each voter; it, then for each voter, encrypts the candidates using the encryption public key and signs each ciphertext with the voter's signing credential. The registrar also includes, for later auditing, on the voting sheet the signing credential and the randomness used to encrypt the candidates. It then sends the voting sheets to the voter.

Voting phase: The voter receives the voting sheet through some channel other than her voting device. She can optionally get her auditing device to check the correctness of the voting sheet. The voter also receives a password from the voting server. The voter, using her voting device authenticates to the voting server using her password. She then scans the signed ciphertext corresponding to her preferred candidate into her voting device. The voting device submits this ciphertext to the voting sever. The voting server checks the validity of the signature and that no ballot had already been received from that voter. If both checks pass the voting server post the ciphertext to the bulletin board and sends an acknowledgment to the voter by way of the voting server.

Tallying phase: The tallying authority decrypts the contents of the bulletin board and publishes the result with appropriate proofs.

3 Verifiability

The authors of BeleniosVS extending on [8, 9] give a pen-and-paper definition of end-to-end verifiability in the symbolic model (Sec. 3.6 of [6]). The informal variant of definition from [6] says that the election result should contain only

- the votes from the voters who successfully performed the verification specified by the protocol;
- some of the votes from the voters who did not make any verification;
- and at most k dishonest votes where k is the number of dishonest voters (under the control of the attacker).

A carefully reader will have noted that the list above makes no mention of what happens to votes from voters who attempted to verify but are unsuccessful, that is for whom the verification checks failed. One would assume that such votes cannot be included in the tally (except as part of the dishonest votes). However, the formal definition given in [6] considers detecting cheating as equivalent to not checking at all and so such voters are put in the second group (which was informally described as not making any verification).

The formal (pen-and-paper) definition is defined using events, which denote the progress of the protocol for each voter.

Voter(id, cred, l): voter *id* is registered with credential *cred*. The label l records if the voter is honest or dishonest.

Voted(id, v): voter *id* has cast a vote for *v*

Goting-to-tally(id, cred, b): ballot *b* has been recorded on the bulletin board by the voting server. According to the voting server, *b* is associated with voter *id* with credential *cred*.

Verified(id, v): voter *id* has voted for *v* and has successful performed all required checks. She should be guaranteed her ballot will be properly counted for *v*.

The description of Verified is misleading because it is used in [6] even for voters who do not audit. We think the use is correct but the description should be updated, since even if the voter does not make any additional checks it still models that their device told them the ballot was correctly recorded. An observant reader will have noted that events provide no way to denote a voter who performs the security checks and detects a problem; for this reason we added a new event. However, due to a technicality in the current model a voter who receives an error during a verification check halts; because of this, at present, all voters who finish have successfully verified. (Future work may wish to resolve this issue but we chose not to because it would require us to change already defined events.)

Finished(id, cred): voter *id* with credential *cred* has finished voting.

3.1 Changes to the ProVerif model

In this section we will discuss our changes to the model which are motivated by the investigation of the ProVerif model in [6] and the attacks that ProVerif finds. In [6, page 379] Cortier et al. write "*in all cases...[which are marked as insecure]..., we found a real attack.*" The statement is completely accurate but at least for us was confusing. The statement means that there are attacks against BeleniosVS in the ProVerif model but it does not mean that the attacks would go undetected in reality. In other words, we initially incorrectly interpreted the statement to mean that the authors **validated** the model by checking the attacks found by ProVerif were attacks on the real system. In fact, their ProVerif model finds various spurious attacks and we make several refinements to the model to remove them. We stress that though the attacks are spurious the system (without changes) is still insecure in the relevant threat models. We have checked that all threat models in which ProVerif finds an attack in our enhanced model have attacks against the real system. It is important to remove these spurious attacks so that ProVerif can finds attacks that would work on the deployed scheme. Once ProVerif found the useful attacks we were able to update the scheme with countermeasures and have ProVerif validate that the countermeasures worked.

Our more complicated security conditions and scheme require us to use more recent features of ProVerif which have only been added after the ProVerif version used in [6]. Even so when considering cases where the register is corrupt, and hence, the public credential may not be unique, we had to assume an extra axiom to make it terminate. The axiom states that only one ballot per name is added

to the ballot box (if the server is honest) which is true since the server enforces the condition (Refinement 4). (It might be possible for ProVerif to check this property and so remove the axiom but we were unable to do so).

One of the difficulties we faced in this work was deciding how to interpret various ambiguities in the original paper [6]. We have tried to make use of the ProVerif files to clarify the authors' intent but with somewhat limited success; the models of the different properties in ProVerif capture slightly different variants of BeleniosVS. For example, the comments in the verifiability model suggest the channel between the voting device and voting server is authenticated but not encrypted. Whereas the receipt-freeness definition assumes the channel is secret. We have attempted to unify these to the largest extent possible.

What does it mean to verify and whose voter is it anyway? In e-voting we tend to think of verification by the voter as Boolean, the voter either does or does not (choose to) verify (her ballot). To some extent, in the original ProVerif analysis of BeleniosVS analysis, and to a greater extent in our refinement, such verification is a sliding scale which captures different actions in different threat models. Even a voter who takes no explicit verification steps is still modelled as believing her intended vote was cast as intended (unless her computer tells her something went wrong). So, even for a voter who does not take any extra verification steps, her device (if honest) will report an error unless all of the checks the device performs pass.

It is standard in the analysis of e-voting voting schemes to split voters into honest and dishonest. However, in our refined threat model of BeleniosVS, we are confronted with voters whose credentials are completely leaked but who make no attempt to cooperate with the adversary; should they be modelled as honest or dishonest? The issue is more nuanced than it might first appear and we adopt the approach of considering voters honest if they attempt to vote (and complain if they fail) but dishonest if they do not attempt to vote themselves or do not complain when voting fails. In doing so we have eliminated the earlier category of "voters who did not make any verification", all voters either attempt to vote and report any error raised by their device, or are considered dishonest.

3.2 The sufficient conditions were not necessary.

As we have noted, the description of Verifiability in Sec. 3.6 of BeleniosVS paper [6] does not explicitly say that the properties do not have to hold if verifiability checks fail. Indeed, the ProVerif definition of cast-as-intended requires the properties to hold even if verification fails.

Refinement 1 *We changed the ProVerif cast-as-intended definition so that the condition need only hold if the voter doesn't complain.*

In the ProVerif security model for BeleniosVS each property has two variants, one which identifies the voter by name and another which identifies the voter based on credential. This is necessary because names are only a reliable

identification if the voting server is honest, and analogously credentials are only reliable if the registrar is honest. We updated both the cast-as-intended definitions but for brevity only describe the change on the name based definition since they are nearly identical.

Cast as intended (ID-based)-Old: For every ballot in the tally, belonging to name n for choice v either the voter n is dishonest or the voter is honest and intended to cast a ballot for v. (We denote by $\diamond$ variables which may take any value.)

$$\text{GOING\_TO\_TALLY}(n, \diamond, \diamond, v) \rightarrow$$
$$(\text{VOTER}(n, \diamond, C)) \vee (\text{VOTER}(n, \diamond, H) \wedge \text{VOTE}(n, v)).$$

Our updated variant is nearly identically but adds the requirement that the voter has not complained.

Cast as intended (ID-based)-New: For every ballot in the tally, belonging to name n for choice v either the voter n is dishonest or the voter is honest, believes their ballot to have been correctly cast and intended to cast a ballot for v. (We denote by $\diamond$ variables which may take any value.)[3]

$$\text{GOING\_TO\_TALLY}(n, \diamond, \diamond, v) \wedge \text{FINISHED}(n, \diamond) \rightarrow$$
$$(\text{VOTER}(n, \diamond, C)) \vee (\text{VOTER}(n, \diamond, H) \wedge \text{VOTE}(n, v)).$$

Refinement 2 *We split the bidirectional channel between the voting device to the voting server into separate channels in each direction. This allows us to refine the adversary's control over the channels. In particular, we refine the case where the voter's password is leaked so that the adversary can impersonate the voter but not the voting server.*

While the first two refinements prevent certain attacks, they do not actually improve the security of the scheme in any threat model since the adversary can perform slightly more complicated attacks to break verifiability. Nevertheless, removing these simple attacks is necessary so that ProVerif identifies the more complicated attacks.

4 Changes to the scheme

In addition to our changes to the model we also made two changes to the scheme.

Refinement 3 *The voting device sends a random nonce to the server along with the vote. The voting server sends this nonce back with the acknowledgement. An honest device only tells the voter that her ballot was cast if the nonce received with the acknowledgement matches the nonce sent.*

[3] Intuition would be better served by using VERIFIED in place of FINISHED in the correspondence assertion below, however for technical reasons this is not possible. Since in our situation FINISHED implies VERIFIED for honest voters, there is no technical issue.

We have deliberately kept this refinement simple to increase deniability; interestingly neither the paper introducing BeleniosRF [7] or BeleniosVS [6] describe in detail how the voter should mislead the coercer.[4] The formal security definitions for both do not capture what the adversary learns by demanding the acknowledgment from the voters. Intuitively, both schemes have good coercion resistance but various nuances are not captured by the formal definition. Intuition also suggest that our refinement does not worsen the situation since the nonce is chosen uniformly, randomly and independently (of the ballot).

The nonce returned to the device is the one submitted with the accepted ballot, since assuming the voting server and device are honest the adversary cannot fake the acknowledgment. However, one might wonder if the adversary can learn the nonce and then submit a different ballot with this nonce and hence trick the voter. We have proved in ProVerif that the adversary can (when both the voting server and voting device are honest) only learn the nonce after the ballot is received by the voting server (at which point no further ballot will be accepted). The ultimate validation of *the refinement* is that it *ensures verifiability* in scenarios which are otherwise vulnerable to attack.

Refinement 4 *The voting server keeps a list of the names of those who have voted. It will not accept more than one vote from each name.*

The original scheme ensured that only one ballot was accepted per credential but allowed multiple votes to come from the same name (though this should never occur if the registrar and voting server are honest). We extend the scheme to also prevent multiple ballots being accepted for the same name. This refinement (and the accompanying axiom we mentioned earlier) are necessary for ProVerif to terminate in two cases where the registrar is dishonest. So far as we can tell this is an artefact of the proof; that is we know of no attack, in an otherwise secure threat model, which is prevented by this refinement. Nevertheless, the refinement seems eminently sensible and we recommend its use in practice.

5 Conclusion Key Takeaways

Based on our analysis and investigation we draw the following two conclusions; the first is important for automatic machine-checked analysis and the second for all e-voting schemes.

Model validation Our work highlights the importance of validating the attacks found by automatic theorem provers; it is always important to check that they work on the deployed scheme.

Defence in depth Our work highlights the importance of basic consistency checks in the honest protocol.

[4] There is some informal discussion but it is unclear what threat model is trying to be captured.

Distinction between verifiability and eligibility verification The definition of "verifiability" in [6] being closer to integrity had the virtue of capturing ballot stuffing attacks which our new definition does not; our new definition is satisfied even if it is possible to cast ballots on behalf of honest voters, who don't vote, without knowing their credential. Our new definition would need to be used alongside a definition of eligibility verification which prevents this attack. Future definitions may wish to consider introducing a special category for honest voters, with leaked credentials, who do not vote.

We show in table 1 a summary of the ProVerif analysis. We have highlighted our improvements in blue. We list the assumptions on participants and information with blank denoting honest, ⧫ denoting dishonest (or leaked), and - denoting none. R denotes the registrar, VS the voting server, VD the voting device, AD the auditing Device, CS the code sheet, P the password. We also include an example attack where relevant.

Dishonest parties and leaked data						Properties	Attack
R	VS	VD	AD	CS	P	Verifiability	
						✓	NA
⧫				⧫		✓	NA
			⧫	⧫		✓	NA
			-	⧫		✓	NA
	⧫	⧫		⧫		✓	NA
	⧫	⧫	-	⧫		✓	NA
				-	⧫	✓	NA
⧫	⧫					×	Casts a different vote
⧫		⧫				×	Casts a different vote
⧫				⧫		×	Bad voting sheet
⧫				-		×	Bad voting sheet
⧫					⧫	✓	NA
⧫				⧫	⧫	✓	NA
	⧫		⧫			×	Casts a different vote
	⧫			⧫		×	Casts a different vote
		⧫	⧫			×	Casts a different vote
		⧫		⧫		×	Casts a different vote
		⧫			⧫	✓	NA
			⧫		⧫	✓	NA
			⧫	⧫	⧫	✓	NA
				⧫	⧫	✓	NA
			-	⧫	⧫	✓	NA

Table 1. Security model

Future work In most of the remaining threat models the adversary can vote on the voter's behalf and this goes undetected because either the voting device or

voting server is dishonest, there are also eight threat models where the registrar can break verifiability for voters who do not audit their code sheet. These lines of attack seem impossible to prevent without introducing new players (or new channels between existing players). Future work, particularly if it has a concrete deployment situation in mind, should revisit if it is feasible to introduce the new players or channels. For example, the voting server could send an SMS to the voter acknowledging that her ballot was received.

As we have noted, the formal definition of "verifiability" in [6] is intuitively much closer to integrity (than verifiability) since it does not model verification checks that fail. We leave as future work updating the formal definition of end-to-end "verifiability" in the symbolic model to catch verification failures; we, also, leave as future work formally showing that our refined sufficient conditions satisfy the (as yet nonexistent new) formal definition.

Source code The ProVerif source files are available at https://github.com/gerlion/Improved-Verifiability-for-BeleniosVS.

Acknowledgments

Thomas Haines was supported by Research Council of Norway and the Luxembourg National Research Fund (FNR), under the joint INTER project SURCVS (INTER/RCN/17/11747298/SURCVS/Ryan).

References

1. B. Adida. Helios: Web-based open-audit voting. In *In Proceedings of the 17th USENIX Security Symposium (Security '08*, 2008.
2. G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub. Easycrypt: A tutorial. In *FOSAD*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73. ACM, 1993.
4. Y. Bertot, P. Castéran, G. Huet, and C. Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, 2004.
5. B. Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Found. Trends Priv. Secur.*, 1(1-2):1–135, 2016.
6. V. Cortier, A. Filipiak, and J. Lallemand. Belenios-VS: Secrecy and verifiability against a corrupted voting device. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 367–36714. IEEE, 2019.
7. V. Cortier, G. Fuchsbauer, and D. Galindo. Beleniosrf: A strongly receipt-free electronic voting scheme. *IACR Cryptology ePrint Archive*, 2015:629, 2015.
8. V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Election verifiability for helios under weaker trust assumptions. In *ESORICS (2)*, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2014.
9. V. Cortier, D. Galindo, and M. Turuani. A formal analysis of the neuchatel e-voting protocol. In *EuroS&P*, pages 430–442. IEEE, 2018.

10. V. Cortier, P. Gaudry, and S. Glondu. Belenios: A simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, volume 11565 of *Lecture Notes in Computer Science*, pages 214–238. Springer, 2019.
11. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–207, 1983.