Cast-as-intended: A formal definition and case studies

Peter B. Rønne, Peter Y. A. Ryan, and Ben Smyth

Interdisciplinary Centre for Security, Reliability and Trust,
University of Luxembourg, Luxembourg

Abstract. Verifiable voting systems allow voters to check whether their ballot is correctly recorded (individual verifiability) and allow anyone to check whether votes expressed in recorded ballots are correctly counted (universal verifiability). This suffices to ensure that honest voters' votes are correctly counted, assuming ballots are properly generated.
Achieving ballot assurance, i.e., assuring each voter that their vote is correctly encoded inside their ballot, whilst ensuring privacy, is a challenging aspect of voting system design. This assurance property is known as cast-as-intended. Unlike many properties of voting systems, it has yet to be formalised. We provide the first formal definition and apply our definition to MarkPledge, Prêt à Voter, Selene, ThreeBallot, and schemes based upon Benaloh challenges.

1 Introduction

End-to-end Verifiable (E2E V) voting systems produce evidence of correct operation: Cast ballots (which contain the vote in encrypted or encoded form) are published on a public bulletin board (ledger), so voters can check whether their ballot is collected (individual verifiability), and tallies are coupled with proofs, so anyone can check whether votes expressed in collected ballots are correctly counted (universal verifiability). By additionally providing means to check whether votes are correctly encoded in ballots, we achieve end-to-end verifiability. This is not yet enough to guarantee that the outcome is correct, for this we need extra measures such as eligibility verifiablity to prevent ballot stuffing and clash attacks [1] etc., but this is beyond the scope of this paper.

For many end-to-end verifiability systems (but not all, e.g., [29, 30]), ballots are constructed using cryptographic operations, which are typically beyond the mathematical capabilities of even the most studious scholar. Moreover, assurance of correct encoding has to be provided without undermining privacy. Arguably, providing ballot assurance, in the face of coercion threats, is is the most challenging aspect of End-to-end verifiability. Nonetheless, various ingenious methods have been devised to provide voters with suitable, non-transferable assurance.

One notion of checking whether ballots correctly encode votes is known as *cast-as-intended* (aka *ballot assurance*).

[1] where more than one voter gets assigned the same ballot

– *Cast-as-intended.* A voter can check whether their ballot correctly expresses their vote.

Discussion of cast-as-intended originates from Chaum [7,8] and was taken further by Neff [27], Adida & Neff [1], and Benaloh [3, 4]. Yet, formal study of cast-as-intended is limited: Unlike other properties of voting systems, a definition of cast-as-intended has yet to be formalised. (Cf. the rich literature on formal definitions of verifiability [10, 18, 20–22, 24, 36, 38, 39] and privacy [5, 6, 11, 13, 15, 16, 20, 21, 25, 26, 34, 37, 40].) Here, we formalise cast-as-intended as a game in the computational model of cryptography, thus filling a gap in the literature.

In this paper we consider two categories of End-to-End Verifiable voting system: *conventional*, wherein each voter checks that a *protected ballot* [2] that represents their vote, typically an encryption of their vote, appears on the bulletin board, and *tracker-based*, where each voter checks their (plaintext) vote appears in the election outcome, using a private tracker. Examples of the former include Prêt à Voter, Helios, and Belenios. Examples of the latter, Selene [32] and sElect [23]. Cast-as-intended is achieved very differently across these categories, so we apply our definition to examples in each. In the first category, voters cast ballots that are encryptions of their votes and tallies should correspond votes embedded in those ballots. Since voters cannot compute ciphertexts, they rely on a device or process to encrypt. Assuring the voter that the resulting ballot correctly encodes their vote without undermining privacy, and in a manner that is usable and understandable, is highly non-trivial, and various techniques have been presented in the literature.

The second category provides verifiability in a more direct and transparent fashion: the voter is able to identify their (plaintext) vote in the tally. In fact, such systems provide *tallied as intended* verification and, in contrast to the conventional schemes, cast as intended is not strictly necessary. Some subtleties nonetheless remain. For example, each voter should be assured that their tracker is unique. We discuss the details later.

For conventional voting systems, in contrast to tracker-based schemes, it is essential that the voter be able to verify that their vote is correctly embedded in the ballot. This is typically achieved via some form of cut-and-choose mechanism: the device is required to commit to one or more encryptions of the voter's vote and all but one of these are randomly selected then audited to confirm that the correct vote was encoded. The remaining, non-audited ballot can then be cast.

Benaloh proposed a mechanism, in effect a sequential cut-and-choose, to provide such assurance [3, §4.2]: A voter inputs a vote into a device, the device computes an encryption of the vote, and commits to the resulting ciphertext, by printing the ciphertext or digitally signing it, for instance. Next, the voter chooses to audit or cast the ciphertext. For the former, if auditing fails, the voter should raise an alarm, otherwise (auditing succeeds), the audit or cast process repeats, until the voter chooses to cast.

Auditing typically reveals coins used to construct ciphertexts, enabling ciphertext reconstruction, which suffices to convince a voter that their ballot ex-

[2] to use Rivest's terminology

presses their vote, assuming they can reconstruct the ciphertext themselves (using a system they trust) or a trusted third party can. Thus, cast-as-intended can be unconditionally achieved, assuming perfect correctness of the underlying encryption scheme.

Reliance on a trusted system or trusted third party to perform ballot audits may seem disingenuous; after all, trust is contrary to the goal of verifiability. However, ciphertexts can be reconstructed by multiple systems, third parties, or both, thereby removing the need to trust any individual device or service.

Most schemes that reveal coins do not allow audited ballots to be cast, to avoid compromising receipt-freeness. (A notable exception is Neff's Mark-Pledge [28].) A consequence of this observation is that ballot assurance is probabilistic rather than deterministic.

Sidebar 1 Preliminaries: Games and notation

Games are probabilistic algorithms that output booleans. An adversary *wins* a game by causing it to output true ($\top$) and the adversary's *success* in a game $\mathsf{Exp}(\cdot)$, denoted $\mathsf{Succ}(\mathsf{Exp}(\cdot))$, is the probability that the adversary wins, i.e., $\mathsf{Succ}(\mathsf{Exp}(\cdot)) = \Pr[\mathsf{Exp}(\cdot) = \top]$. Adversaries are *stateful*, i.e., information persists across invocations of an adversary in a game.

We let $A(x_1, \ldots, x_n; r)$ denote the output of probabilistic algorithm A on inputs $x_1, \ldots, x_n$ and coins r, and we let $A(x_1, \ldots, x_n)$ denote $A(x_1, \ldots, x_n; r)$, where coins r are chosen uniformly at random from the coin space of algorithm A. Moreover, we let $x \leftarrow T$ denote assignment of T to x, and we write $(x_1, \ldots, x_{|T|}) \leftarrow T$ for $x \leftarrow T$; $x_1 \leftarrow x[1]; \ldots; x_{|T|} \leftarrow x[|T|]$, when T is a vector.

2 Security definition

For schemes in the conventional category, cast-as-intended requires ballots be correctly constructed. For many schemes, correct ballot construction simply requires computing a valid encryption of the vote, with respect to the public key of the tabulation process. For universally-verifiable voting systems, it follows that if a voter's ballot correctly encodes the voter's vote, and that ballot is correctly tallied, then the voter's vote will be correctly included in the tally.

We formalise cast-as-intended in the syntax proposed by Smyth, Frink & Clarkson [38]: Construction is defined by a probabilistic polynomial-time algorithm Vote, wherein

> Vote takes as input a public key pk, a voter's vote v, some number of candidates nc, and a security parameter κ, and outputs a ballot b or error symbol $\perp$, where vote v should be selected from a sequence $1, \ldots, nc$ of candidates.

Universal verifiability assures us that votes expressed in ballots constructed using algorithm Vote will be correctly counted, and cast-as-intended enables voters to check that their vote is expressed in a ballot constructed using algorithm Vote.

Ballot construction differs between systems. In some cases, the ballot may be generated by software on the voter's computer. Other cases may be more elaborate, e.g., Prêt à Voter involves authorities generating and distributing blank two-column paper ballots, and voters physically marking their selections before discarding the left column. Yet further systems (e.g., code-voting, PunchScan, Scantegrity, and PGD) encode votes by means other than encryption, using material from an initialisation phase. For example, in code voting, a correspondence between vote options and codes is committed in print on the code sheets, and this should be consistent with the mapping of codes to votes that is committed to the bulletin board. Some verifiable schemes even avoid cryptography, e.g., Randell/Ryan [29] and ThreeBallot [30].

Formalising cast-as-intended involves defining what it means for a ballot to be correctly constructed and the means to verify this, which can be captured as a game that challenges an adversary to achieve the opposite: To dupe a voter into believing their ballot encodes their vote, when it does not. We model ballot construction as an algorithm $\mathcal{A}$ parametrised by a public key pk, some number of candidates nc, and a security parameter κ, and we model the voter as an algorithm $\mathcal{V}$ that takes a vote v and a distinct security parameter $\hat{\kappa}$ as input. The latter security parameter determines the probability of breaking cast-as-intended, whereas the former determines the probability of breaking other security properties, such as individual and universal verifiability. We use distinct security parameters, because the complexity of algorithm $\mathcal{V}$ should be upper-bounded by the capability of a voter's mind, whereas algorithm $\mathcal{A}$ should be computable by machine.

The system is assumed to be adversarial and, to achieve cast-as-intended, some trustworthy device is also required. (Alternatively, a multitude of devices may be used, only one of which need be trustworthy.) We model such a device as an algorithm $\mathcal{T}$ that takes the same parameters as algorithm $\mathcal{A}$. Hence, ballot construction is modelled by the computation $b \leftarrow \mathcal{V}^{\mathcal{T},\mathcal{A}}(v, \hat{\kappa})$, where algorithm $\mathcal{T}$ is parameterised by pk, nc, and κ. If cast-as-intended is achieved, then such computations should result in a ballot $b = \mathsf{Vote}(pk, v, nc, \kappa; r)$ for some coins r or the error symbol $\perp$ (representing a voter detecting malice), which leads to to our security definition (Definition 1). In that definition, we extend algorithm $\mathcal{A}$ with the capabilities of a malicious administrator that defines the public key, the voter's vote, and the number of candidates, when parametrised by security parameters κ and $\hat{\kappa}$.

Definition 1 (Cast-as-intended). *Let $\mathcal{V}$, $\mathcal{T}$, Vote, and $\mathcal{A}$ be probabilistic polynomial-time algorithms, κ and $\hat{\kappa}$ be security parameters, and* Cast-As-Intended *be the following game.*

$\mathsf{Cast\text{-}As\text{-}Intended}(\mathcal{V}, \mathcal{T}, \mathsf{Vote}, \mathcal{A}, \kappa, \hat{\kappa}) =$

$(pk, v, nc) \leftarrow \mathcal{A}(\kappa, \hat{\kappa})$;
$b \leftarrow \mathcal{V}^{\mathcal{T}(pk,nc,\kappa),\mathcal{A}}(v, \hat{\kappa})$;
return $\forall r \,.\, b \neq \mathsf{Vote}(pk, v, nc, \kappa; r) \wedge b \neq \perp \wedge 1 \leq v \leq nc$;

We say $\mathcal{V}, \mathcal{T}, \mathsf{Vote}$ *satisfies* $\delta(\hat{\kappa})$-cast-as-intended*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a function* δ *and for all security parameters* κ *and* $\hat{\kappa}$*, we have* $\mathsf{Succ}(\mathsf{Cast\text{-}As\text{-}Intended}(\mathcal{V}, \mathcal{T}, \mathsf{Vote}, \mathcal{A}, \kappa, \hat{\kappa})) \leq \delta(\hat{\kappa}) + \mathsf{negl}(\kappa)$.

An adversary $\mathcal{A}$ that wins Cast-As-Intended is able to identify a strategy, including choosing a public key pk, a vote v, and a number of candidates nc, such that a voter $\mathcal{V}$ will be deceived into casting a ballot b that does not express their vote v. That is, winning signifies an attack against cast-as-intended.

An election scheme satisfying cast-as-intended guarantees that a voter can check whether their ballot is an output of algorithm Vote, parametrised with their vote v along with public key pk, number of candidates nc, and security parameter κ, for some coins r. Supposing universal verifiability, a voter can check whether the election outcome corresponds to votes expressed in tallied ballots.

Our definition assumes that each voter performs verification steps correctly. Beyond the definition's scope, we must consider the practicality of voters correctly performing verification. Thus, even when a voting system satisfies cast-as-intended, that system may still be vulnerable to attacks arising from voter error or negligence (possibly attacker induced).

Let us finally remark on an important point, namely, that the definition depends crucially on the protocol description for the honest voter $\mathcal{V}$. If we have a protocol that always aborts the casting, then this would reduce the adversary's advantage to zero and we would claim it satisfies cast-as-intended. Hence, we also require a soundness condition asserting that $\mathcal{V}$ should only abort if the ballot is ill-formed, or more exactly, that this should only occur with bounded probability: We assume that $\mathcal{V}$ only aborts when indeed she has detected the b is ill-formed. In practice, whether we can check if a voter is maliciously aborting enters into the territory of *dispute resolution*.

3 Examples

3.1 Prêt à Voter

Fig. 1 Prêt à Voter ballot (a) and receipt (b) for vote "Idefix"

(a)

Obelix	
Idefix	
Asterix	
Panoramix	
	7304944

(b)

X
7304944

In Prêt à Voter [9] the voter gets a ballot with a left and right hand side. The candidates are listed in plaintext in a random order on the left hand side (LHS)

and this order is embedded cryptographically in a value on the right hand side (RHS). In the privacy of the booth, the voter marks their selection on the right hand side against the candidate(s) as indicated on the LHS. Once they have made their mark(s) they detach and destroy the LHS, i.e. removes the plaintext list of candidates, thus concealing the selection and creating the receipt.

The universal verifiability of the scheme will afterwards guarantee that the vote is processed according to the encrypted order. The ballot is thus correctly constructed, if the encrypted candidate order on the RHS matches the one displayed in plaintext on the LHS. To check this the voter can perform ballot audits to verify this, but cannot verify the actual cast ballot. Additionally, ballots can be randomly audited by observers. Note that for Prêt à Voter ballot auditing can be delegated and is entirely privacy preserving and indeed dispute resolving.

Let us first consider the case where voters are instructed to choose $\hat{\kappa}$ ballots and audit all of these but one. In a worst case scenario, the adversary is able to inject a chosen number of maliciously created ballots into this selection $\hat{\kappa}$ ballots. The adversary obtains the best advantage by injecting precisely one ballot and wins if this is used for vote-casting, i.e. $P_{\text{Adv succ}} = 1/\hat{\kappa}$, i.e. $\delta \geq 1/\hat{\kappa}$.

We can also consider the case, where a voter audits a ballot with a probability p. We further assume that the probability that a ballot is audited by observers is q. We then find

$$P_{\text{Adv succ}} = (1 - q) \cdot (1 - p).$$

Using the arguments for Benaloh challenges below, this also holds if the voter audits multiple times with the same probability.

Here we are assuming that the random audits are unpredictable and that chain-of-custody of the set of ballot forms is guaranteed, i.e. fake ballot forms cannot be injected after the observer audits are performed. In fact, observer audits can be performed before, during and after voting, so this assumption is quite mild.

3.2 Benaloh

Let us now make a simplified analysis of Benaloh challenges. We ignore all information that could be leaked by the vote choice used by the voter. Let us assume that the voter has fixed probability in each round to audit. We can argue that this makes sense since the voter obtains no other knowledge during the Benaloh challenge process to influence the probability, and we ignore vote choices. For the adversary, we will also assume he has a fixed probability in each round. Let p_i denote the probability of the voter doing an audit in round i and q_i the probability that the adversary changes a vote in this round.

The probability that the adversary is successful is obtained by sum of the probabilities that he is successful in round i which is the probability that the voter challenges all earlier rounds, that the adversary was honest in all of these,

and then he cheated in round i. That is

$$P_{\text{Adv succ}} = \sum_{i=1}^{\infty} p_1 \cdots p_{i-1}(1-q_1)\cdots(1-q_{i-1})(1-p_i)q_i$$

.

We only aim to make a simplified analysis and we do this by assuming that the probabilities will not depend on the rounds. For the voter this could happen if we specify to the voter to make an audit with a certain probability. Then we get

$$P_{\text{Adv succ}} = \sum_{i=1}^{\infty} p^{i-1}(1-q)^{i-1}(1-p)q = \frac{(1-p)q}{1-p(1-q)}.$$

This function is increasing in cheat probability q. Thus if the adversary wants to maximise this, he chooses $q = 1$ i.e. to always cheat. The winning probability is then

$$P_{\text{Adv succ}} = 1 - p,$$

corresponding to the probability that the voter casts in the first round. That is we have $\delta \geq 1-p$. Note that the average number of challenge rounds in the vote casting is $1/(1-p)$ (without an adversary present). Abusing notation, we thus have that $\delta \geq 1/\hat{\kappa}$ where $\hat{\kappa}$ is the number of challenge rounds.

Note however, that if the voter is not prescribed a certain audit probability p and doesn't care about the effort of doing the audits, then $P_{\text{Adv succ}}$ is a decreasing function in p i.e. the voter would choose $p \approx 1$, i.e. to almost always audit. A full analysis should be done via game theory, see also [12].

3.3 MarkPledge

MarkPledge, [27], is of particular interest in the context of cast-as-intended as it provides assurance of correctness of the ballot that is actually cast. This is in contrast to cut-and-choose style ballot assurance, e.g. Benaloh challenges, where an audited ballot cannot be cast.

At a very high level, MarkPlegde involves the voter interacting with a device in the booth to perform an interactive zero-knowledge proof of correctness of the ballot. In essence, this proof serves to convince the voter that a 1 is encrypted against the candidate of choice and a transcript of this proof is printed on the receipt. As part of this interactive ZK proof, the voter provides a random challenge, e.g. a string of k digits.

This alone would not of course be receipt-free, so the chosen vote is masked by the device constructing fake ZK proof transcripts that an encryption of 1 against the other candidates. The device also provides a ZK proof that there is exactly one encryption of 1 and the other encryptions are of 0. Anyone later seeing the receipt cannot distinguish the real and fake proofs and hence cannot identify which candidate was selected. Only the voter will know for which candidate they executed the real, interactive proof.

During tabulation, the candidates with an encryption of zero are weeded out, exploiting the homomorphic properties of the encryption, leaving just those with the encryption of 1.

The ballot assurance provided by MarkPledge is therefore dependent on the size of the challenge space. So if the challenges are strings of k^* digits, we get a bound on the chance of the device being about to cheat the voter of $p = 1/10^{k^*}$.

3.4 ThreeBallot

In the ThreeBallot scheme by Rivest [30] a voter distributes their vote intent over three ballots. The scheme is interesting in this context as it does not employ cryptography, and voter can tell directly whether the (three)ballot is correctly constructed. We will also assume here that mechanisms are in place to ensure that the (three)ballot is well formed, i.e. obeys the rules (two votes for the chosen candidate and one for the others).

The ballots contain unique serial numbers. These are sent to a Bulletin Board, but the voter chooses one of the single ballots and gets a copy of it. The voter's choice of which of the three ballots should be concealed from the system. At home, the voter checks that the single ballot with the corresponding serial number appears online and that the partial vote choice in that ballot is correctly stored.

Note that this example is special since this verification step is also a stored-as-cast verification. With a malicous authority involved the serial numbers might not be unique, and could give rise to clash attacks. In this case it is not really meaningful to say that the displayed ballot is the voters ballot, since it will be assigned to one or more voters. However, in the scope of cast-as-intended, we we will focus on a single voter setting, and leave such problems to other parts of verifiability.

If we assume that the voter chooses the receipt slip uniformly at random and the adversary is unaware of this choice, then the adversary can try to change one single ballot of three, and will succeed with an undetected change of the ballot with probability 2/3. Thus $\delta \geq 2/3$

Scheme	Prêt à Voter	Benaloh	MarkPledge	ThreeBallot	Selene
δ	$1/\hat{\kappa}_1$	$1/\hat{\kappa}_2$	$1/10^{\hat{\kappa}}$	2/3	0
Delegable	✓	✓	✗	✓	✗

Table 1. The different cast-as-intended protocols and their respective cast-as-intended δ-value. $\hat{\kappa}_1 = (1-p)\cdot(1-q)$, $\hat{\kappa}_2 = (1-p)$. We also note if the verification checks, or parts thereof, are delegable.

4 Tracker-based schemes

Ryan, Rønne, & Iovino recently [32] introduced an orthogonal approach in which the cast-as-intended is first established after tallying. For traditional voting systems, voters perform a cast-as-intended check to ensure their ballot correctly expresses their vote, an individual-verifiability check to ensure their ballot is collected, and a universal-verifiability check to ensure their vote is counted. By comparison, Ryan, Rønne, & Iovino propose that voters simply check whether their plaintext vote is present in the tally, using a private tracker—the standard early cast-as-intended and individual-verifiability checks are not necessary; a more direct, more transparent form of verification is achieved. Voting systems Selene [32] and sElect [23] achieve this new form of tracker-based verifiability, with the main difference being that Selene first releases the tracker to the voter after tallying to provide coercion-mitigation - the voter can equivocate the tracker to a tracker for another vote.

4.1 Example: Selene

Selene, [32] is an example of a tracker-based verifiable scheme in which the voter can confirm directly that their vote is included in the tally. The mechanism can be applied to various forms of e-voting schemes [2, 17, 31, 33], has been studied formally [19, 42], and has been studied from a usability viewpoint [14, 41]. In Selene the voters hold secret trapdoor keys. After the end of election and after the tally has been made public, the voters will receive an cryptographic term, called the alpha term. This is combined with a public beta term which is available on the bulletin board to form an ElGamal encryption of the tracker under the voter's trapdoor key. The voter can now decrypt this to retrieve their tracker. The tracker is then used to check the plaintext vote on the final tally board.

The authorities could send a fake alpha term to the voter, however, as is proven in [32] the chance of such an alpha term opening to an existing tracker is negligible assuming that the authorities do not know the voter's trapdoor key and under a computational assumption (hardness of computing g^{x^2} given g^x without knowing x). Thus if the voter's trapdoor key is not leaked and the computational assumption holds, then δ is negligible. This presupposes that the proofs and verifiable computations on the bulletin board are verified, which can be done by the voter herself, or any third party.

In a tracker based scheme, traditional cast-as-intended and individual- and universal-verifiable checks are all bundled into a single tallied-as-cast check.

5 Outlook

Verifiability has emerged as a means to ensure integrity of elections. Several aspects of verifiability have been identified: well studied notions of individual- and universal-verifiability, and the seemingly less well understood notion of cast-as-intended. We propose the first formal definition of cast-as-intended, closing

a gap in the literature and enabling analysis of systems purporting to achieve cast-as-intended.

Future work could flesh out and make more rigorous the arguments we sketch for MarkPledge, Prêt à Voter, Selene, ThreeBallot, and schemes based upon Benaloh challenges. Other voting systems could also be analysed, e.g., Pretty Good Democracy and Bingo Voting. It would also be interesting to analyse any relation between cast-as-intended and dispute resolution, privacy during ballot construction, or both. Perhaps most importantly: A suitable security notion to bridge the gap between individual verifiability and cast-as-intended should be sought, see also [35] which points out that individual- and universal-verifiability plus cast-as-intended does not yield end-to-end verifiable voting systems, e.g. clash attacks are not captured. It would also be interesting to see to which extent trust can be removed from the cast-as-intended verification [18].

Another interesting future path is to relate the parameter $\hat{\kappa}$ more tightly to usability in order to compare the achieved level of security to the amount and complexity of user-interaction required. We already did a preliminary attempt of this with the values shown in Table 1 where $\hat{\kappa}$ represents the number of interactions or the number of digits a voter needs to handle, but comparing and understanding the usability, especially across systems, will require actual user studies.

Acknowledgements. This work received financial support from the Luxembourg National Research Fund (FNR) under the PolLux/CORE project STV (12685695) and the FNR CORE project EquiVox (13643617).

References

1. Ben Adida and C. Andrew Neff. Ballot casting assurance. In *EVT'06: Electronic Voting Technology Workshop*. USENIX Association, 2006.
2. Mohammed Alsadi and Steve Schneider. Verify my vote: Voter experience. *E-Vote-ID 2020*, page 280, 2020.
3. Josh Benaloh. Simple Verifiable Elections. In *EVT'06: Electronic Voting Technology Workshop*. USENIX Association, 2006.
4. Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In *EVT'07: Electronic Voting Technology Workshop*. USENIX Association, 2007.
5. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *S&P'15: 36th Security and Privacy Symposium*, pages 499–516. IEEE Computer Society, 2015.
6. Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. *Journal of Computer Security*, 26(3):367–422, 2018.
7. David Chaum. Secret-Ballot Receipts and Transparent Integrity: Better and less-costly electronic voting at polling places. `https://web.archive.org/web/*/http://vreceipt.com/article.pdf`, 2002.
8. David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, 2(1):38–47, 2004.

9. David Chaum, Peter YA Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *European Symposium on Research in Computer Security*, pages 118–139. Springer, 2005.
10. Veronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. SoK: Verifiability Notions for E-Voting Protocols. In *S&P'16: 37th IEEE Symposium on Security and Privacy*, pages 779–798. IEEE Computer Society, 2016.
11. Cas Cremers and Lucca Hirschi. Improving Automated Symbolic Analysis for E-voting Protocols: A Method Based on Sufficient Conditions for Ballot Secrecy. arXiv, Report 1709.00194, September 2017.
12. Chris Culnane and Vanessa Teague. Strategies for voter-initiated election audits. In *International Conference on Decision and Game Theory for Security*, pages 235–247. Springer, 2016.
13. Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.
14. Verena Distler, Marie-Laure Zollinger, Carine Lallemand, Peter B. Rønne, Peter Y. A. Ryan, and Vincent Koenig. Security - visible, yet unseen? In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 605. ACM, 2019.
15. Ashley Fraser, Elizabeth A. Quaglia, and Ben Smyth. A critique of game-based definitions of receipt-freeness for voting. In *ProveSec'19: 13th International Conference on Provable and Practical Security*, volume 11821 of *LNCS*, pages 189–205. Springer, 2019.
16. Thomas Haines and Ben Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2020.
17. Vincenzo Iovino, Alfredo Rial, Peter B. Rønne, and Peter Y. A. Ryan. Using selene to verify your vote in JCJ. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *Lecture Notes in Computer Science*, pages 385–403. Springer, 2017.
18. Vincenzo Iovino, Alfredo Rial, Peter B. Rønne, and Peter Y. A. Ryan. Universal unconditional verifiability in e-voting without trusted parties. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 33–48. IEEE, 2020.
19. Wojciech Jamroga, Michal Knapik, and Damian Kurpiewski. Model checking the SELENE e-voting protocol in multi-agent logics. In Robert Krimmer, Melanie Volkamer, Véronique Cortier, Rajeev Goré, Manik Hapsara, Uwe Serdült, and David Duenas-Cid, editors, *Electronic Voting - Third International Joint Conference, E-Vote-ID 2018, Bregenz, Austria, October 2-5, 2018, Proceedings*, volume 11143 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2018.
20. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In David Chaum, Markus Jakobsson, Ronald L. Rivest, and Peter Y. A. Ryan, editors, *Towards Trustworthy Elections: New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 37–63. Springer, 2010.
21. Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In *EUROCRYPT'15: 34th International Confer-*

ence on the Theory and Applications of Cryptographic Techniques, volume 9057 of *LNCS*, pages 468–498. Springer, 2015.
22. Steve Kremer, Mark D. Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10: 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010.
23. Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. select: A lightweight verifiable remote voting system. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 341–354. IEEE, 2016.
24. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and relationship to verifiability. In *CCS'10: 17th ACM Conference on Computer and Communications Security*, pages 526–535. ACM Press, 2010.
25. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. *Journal of Computer Security*, 20(6):709–764, 2012.
26. Tal Moran and Moni Naor. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In *CRYPTO'06: 26th International Cryptology Conference*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.
27. C. Andrew Neff. Practical high certainty intent verification for encrypted votes. Unpublished manuscript, 2004.
28. C Andrew Neff. Practical high certainty intent verification for encrypted votes, 2004.
29. Brian Randell and Peter YA Ryan. Voting technologies and trust. *IEEE Security & Privacy*, 4(5):50–56, 2006.
30. Ronald L Rivest. The threeballot voting system. 2006.
31. Peter B Rønne, Peter YA Ryan, and Marie-Laure Zollinger. Electryo, in-person voting with transparent voter verifiability and eligibility verifiability. *E-Vote-ID 2018*, page 147, 2018.
32. Peter YA Ryan, Peter B Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *International Conference on Financial Cryptography and Data Security*, pages 176–192. Springer, 2016.
33. Muntadher Sallal, Steve Schneider, Matthew Casey, François Dupressoir, Helen Treharne, Constantin Catalin Dragan, Luke Riley, and Phil Wright. Augmenting an internet voting system with selene verifiability using permissioned distributed ledger. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020*, pages 1167–1168. IEEE, 2020.
34. Ben Smyth. Ballot secrecy: Security definition, sufficient conditions, and analysis of Helios. Cryptology ePrint Archive, Report 2015/942, 2018.
35. Ben Smyth. Mind the Gap: Individual- and universal-verifiability plus cast-as-intended don't yield verifiable voting systems. Technical Report 2020/1054, Cryptology ePrint Archive, 2020.
36. Ben Smyth. Surveying global verifiability. *Information Processing Letters*, 163, 2020.
37. Ben Smyth and David Bernhard. Ballot secrecy and ballot independence coincide. In *ESORICS'13: 18th European Symposium on Research in Computer Security*, volume 8134 of *LNCS*, pages 463–480. Springer, 2013.
38. Ben Smyth, Steven Frink, and Michael R. Clarkson. Election Verifiability: Cryptographic Definitions and an Analysis of Helios and JCJ. Cryptology ePrint Archive, Report 2015/233, 2017.

39. Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS'10: Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, volume 6186 of *LNCS*, pages 165–182. Springer, 2010.
40. Dominique Unruh and Jörn Müller-Quade. Universally Composable Incoercibility. In *CRYPTO'10: 30th International Cryptology Conference*, volume 6223 of *LNCS*, pages 411–428. Springer, 2010.
41. Marie-Laure Zollinger, Verena Distler, Peter Roenne, Peter Ryan, Carine Lallemand, and Vincent Koenig. User experience design for e-voting: How mental models align with security mechanisms. 2019.
42. Marie-Laure Zollinger, Peter B. Rønne, and Peter Y. A. Ryan. Short paper: Mechanized proofs of verifiability and privacy in a paper-based e-voting scheme. In Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*, volume 12063 of *Lecture Notes in Computer Science*, pages 310–318. Springer, 2020.