Election Verifiability: Cryptographic Definitions and an Analysis of Helios and JCJ

Ben Smyth
Huawei Technologies Co. Ltd.,
Boulogne-Billancourt, France
research@bensmyth.com

Steven Frink
Cornell University
Ithaca, NY, US
sfrink@cs.cornell.edu

Michael R. Clarkson
Cornell University
Ithaca, NY, US
clarkson@cs.cornell.edu

Abstract—Definitions of election verifiability in the computational model of cryptography are proposed. The definitions formalize notions of voters verifying their own votes, auditors verifying the tally of votes, and auditors verifying that only eligible voters vote. The Helios (Adida et al., 2009) and JCJ (Juels et al., 2010) election schemes are analyzed using these definitions. Helios 4.0 satisfies the definitions, but Helios 2.0 does not because of previously known attacks. JCJ does not satisfy the definitions because of a trust assumption it makes, but it does satisfy a weakened definition. Two previous definitions of verifiability (Juels et al., 2010; Cortier et al., 2014) are shown to permit election schemes vulnerable to attacks, whereas the new definitions prohibit those schemes.

I. INTRODUCTION

Electronic voting systems that have been deployed in real-world, large-scale public elections place extensive trust in software and hardware. Unfortunately, instead of being trustworthy, many systems are vulnerable to attacks that could bring election outcomes into disrepute [19], [45], [56], [84]. So relying solely on trust in voting systems is unwise; verification of election outcomes is essential.

Election verifiability enables voters and auditors to ascertain the correctness of election outcomes, regardless of whether the software and hardware of the voting system are trustworthy [2], [3], [25], [57], [75]. Kremer et al. [63] decompose election verifiability into three aspects:

- **Individual verifiability**: voters can check that their own ballots are recorded.
- **Universal verifiability**: anyone can check that the tally of recorded ballots is computed properly.
- **Eligibility verifiability**: anyone can check that each tallied vote was cast by an authorized voter.

We propose new definitions of these three aspects of verifiability in the computational model of cryptography. We show that individual and universal verifiability are orthogonal, and that eligibility verifiability implies individual verifiability.

Because some electronic voting systems implement voter authentication to third parties, we develop two variants of our definitions—one for systems with internal authentication and another for systems with external authentication.

We employ our definitions to analyze the verifiability of two well-known election schemes, JCJ [59] and Helios [6]. JCJ is an election scheme that achieves coercion resistance and has been implemented as Civitas [29]; it implements its own internal authentication. Helios is a web-based voting system that has been deployed in the real-world and outsources authentication.

The Helios 2.0 election scheme is known to have vulnerabilities that enable attacks on verifiability, and several patches for those vulnerabilities have been proposed [17], [18], [33], [34]. By employing those proposed patches, we obtain a scheme called Helios 4.0 that satisfies our definition of election verifiability with external authentication. Helios 2.0, as expected, fails to satisfy our definition.

The JCJ election scheme does not satisfy our definition of eligibility verifiability, because an adversary who learns the tallyer’s private key could cast unauthorized votes. We introduce a weakened definition of eligibility verifiability, incorporating JCJ’s trust assumption that the private key is unknown to the adversary, and show that JCJ satisfies our weakened definition of election verifiability with internal authentication.

Our definitions of election verifiability improve upon two previous definitions [32], [59] by detecting a new class of collusion attacks, in which the tallying algorithm announces an incorrect tally, and the verification algorithm colludes with the tallying algorithm to accept the incorrect tally. Examples of collusion attacks include vote stuffing, and announcing tallies that are independent of the election. Our definitions also improve upon those previous definitions by detecting a new class of biasing attacks, in which the verification algorithm rejects some legitimate election outcomes. Examples of biasing attacks include rejecting outcomes in which a particular candidate does not win, and rejecting all election outcomes, even correct outcomes.

This paper thus contributes to the security of electronic voting systems by

- proposing computational definitions of election verifiabil-
showing that individual, universal, and eligibility verifiability are mostly orthogonal properties of voting systems, proving that well-known election schemes do (or do not) satisfy election verifiability, and identifying collusion and biasing attacks as new classes of attacks on voting systems and demonstrating that they are not detected by two earlier definitions.

Ours are the first proofs that Helios 4.0 and JCJ satisfy a computational definition of verifiability.

**Structure:** Section II defines election verifiability with external authentication. Section III analyzes Helios. Section IV introduces collusion and biasing attacks. Section V analyzes JCJ. Section VI introduces collusion and biasing attacks. Section VII reviews related work, and Section VIII concludes.

II. EXTERNAL AUTHENTICATION

Some election schemes do not implement authentication themselves, but instead rely on an external authentication mechanism. Helios, for example, supports authentication with Facebook, Google and Yahoo credentials. In essence, the election scheme outsources ballot authentication. We begin by defining election verifiability for that model.

A. Election scheme

An election scheme with external authentication, which henceforth in this section we abbreviate as “election scheme,” is a tuple (Setup, Vote, Tally, Verify) of probabilistic polynomial-time (PPT) algorithms:

- **Setup,** denoted \( (PK_T, SK_T, m_B, m_C) \) \( \xrightarrow{} \) Setup\( (k) \), is executed by the tallier, who is responsible for tallying ballots. Setup takes a security parameter \( k \) as input and outputs a key pair \( (PK_T, SK_T) \), a maximum number of ballots \( m_B \), and a maximum number of candidates \( m_C \).

- **Vote,** denoted \( b \xleftarrow{} \) Vote\( (PK_T, n_C, \beta, k) \), is executed by voters. A voter makes a choice of candidate from a sequence \( \beta_1, \ldots, \beta_{n_C} \) of candidates. A well-formed choice is an integer \( \beta \), such that \( 1 \leq \beta \leq n_C \). Vote takes as input the public key \( PK_T \) of the tallier, the number \( n_C \) of candidates, the voter’s choice \( \beta \) of candidate, and security parameter \( k \). It outputs a ballot \( b \), or error symbol \( \bot \). An error may occur if the choice of candidate is not well-formed or for other reasons particular to the election scheme.

- **Tally,** denoted \( (X, P) \xleftarrow{} \) Tally\( (PK_T, SK_T, BB, n_C, k) \), is executed by the tallier. It involves a public bulletin board \( BB \), which we model as a set of ballots. Tally takes as input the public key \( PK_T \) and private key \( SK_T \) of the tallier, the bulletin board \( BB \), the number of candidates \( n_C \), and security parameter \( k \). It outputs a tally \( X \) and a non-interactive proof \( P \) that the tally is correct. A tally is a vector \( X \) of length \( n_C \) such that \( X[j] \) indicates the number of votes for candidate \( c_j \).

- **Verify,** denoted \( v \xleftarrow{} \) Verify\( (PK_T, BB, n_C, X, P, k) \), can be executed by anyone to audit the election. Verify takes as input the public key \( PK_T \) of the tallier, the bulletin board \( BB \), the number of candidates \( n_C \), a tally \( X \), a proof \( P \) of correct tallying, and security parameter \( k \). It outputs a bit \( v \), which is 1 if the tally successfully verifies and 0 otherwise. We assume that Verify is deterministic.

Election schemes must satisfy Correctness, which asserts that tallies produced by Tally corresponds to the choices input to Vote:

**Definition 1** (Correctness). There exists a negligible function \( \mu \), such that for all security parameters \( k \), integers \( n_B \) and \( n_C \), and choices \( \beta_1, \ldots, \beta_{n_B} \in \{1, \ldots, n_C\} \), it holds that if \( Y \) is a vector of length \( n_C \) whose components are all 0, then

\[
\Pr[(PK_T, SK_T, m_B, m_C) \xleftarrow{} \text{Setup}(k); \quad \text{for } 1 \leq i \leq n_B \text{ do } \quad b_i \xleftarrow{} \text{Vote}(PK_T, n_C, \beta_i, k); \quad Y[\beta_i] \xleftarrow{} Y[\beta_i] + 1; \quad BB \xleftarrow{} \{b_1, \ldots, b_n\}; \quad (X, P) \xleftarrow{} \text{Tally}(PK_T, SK_T, BB, n_C, k); \quad n_B \leq m_B \land n_C \leq m_C \Rightarrow X = Y > 1 - \mu(k)].
\]

Note that Correctness does not involve an adversary. Correctness therefore stipulates that, under ideal conditions, an election scheme does indeed produce the correct tally. Correctness is not actually necessary to achieve verifiability: our definition of universal verifiability will ensure that, in the presence of an adversary, Verify detects any errors in the tally. But it is reasonable to rule out election schemes that simply do not work properly under ideal conditions.

Election schemes must also satisfy Completeness, which stipulates that tallies produced by Tally will actually be accepted by Verify:

**Definition 2** (Completeness). There exists a negligible function \( \mu \), such that for all security parameters \( k \), bulletin boards \( BB \), and integers \( n_C \), it holds that

\[
\Pr[(PK_T, SK_T, m_B, m_C) \xleftarrow{} \text{Setup}(k); \quad (X, P) \xleftarrow{} \text{Tally}(PK_T, SK_T, BB, n_C, k); \quad |BB| \leq m_B \land n_C \leq m_C \Rightarrow \text{Verify}(PK_T, BB, n_C, X, P, k) = 1] > 1 - \mu(k)].
\]

Bulletin boards have also been modeled as public broadcast channels [35], [36], [37]. We abstract from the details of channels by employing sets to represent the data sent on them. We favor sets over multisets, because Cortier and Smyth [33], [34] demonstrate attacks against privacy when the bulletin board is modeled as a multiset.

\(^3\)Let \( X[i] \) denote component \( i \) of vector \( X \).
Without Completeness, election schemes might be vulnerable to biasing attacks, as we show in Section [VI-B].

Finally, election schemes must satisfy Injectivity, which asserts that a ballot cannot be interpreted as a vote for more than one candidate:

**Definition 3 (Injectivity).** For all security parameters \( k \), public keys \( PK_\tau \), integers \( nC \), and choices \( \beta \) and \( \beta' \), such that \( \beta \neq \beta' \), we have

\[
\Pr[b \leftarrow \text{Vote}(PK_\tau, nC, \beta, k); \quad b' \leftarrow \text{Vote}(PK_\tau, nC, \beta', k) : \quad b \neq \perp \land b' \neq \perp \Rightarrow b \neq b'] = 1. \]

Injectivity ensures that distinct choices are not mapped by Vote to the same ballot. Without Injectivity, an election scheme might produce ballots whose meaning is ambiguous. For example, if \( \text{Vote}(PK_\tau, nC, \beta, k; r) \) were defined to be \( \beta + r \), then a ballot \( b \) could be tallied as any well-formed choice \( \beta' \) such that \( \beta' = b - r' \) for some \( r' \). But that definition of Vote is prohibited by Injectivity. Thus, Injectivity helps to ensure that the choices used to construct ballots can be uniquely tallied.

**Limitations:** Our model of election schemes is sufficient to analyze Helios and, after we extend the model to handle internal authentication in Section [IV-A] JCJ. These are notable schemes, and formally analyzing their verifiability is a novel contribution. But there are other notable schemes that fall outside our model:

- Pret à Voter [25], MarkPledge [73], Scantegrity II [24], and Remotegrity [85] all rely on features implemented with paper, such as scratch-off surfaces and detachable columns.
- Everlasting privacy [72], which requires Vote to output a public ballot and a secret proof, involving temporal information, to the voter.
- Scytl’s Pnyx.core ODBP 1.0 [28], which requires the bulletin board to be divided into two parts: a public part visible to all participants, and a secret part visible only to election administrators.

We leave extension of our model to other election schemes as future work.

### B. Election verifiability

Election verifiability comprises three aspects: individual, universal, and eligibility verifiability. We express each as an experiment, which is an algorithm that outputs 0 or 1. The adversary wins an experiment by causing it to output 1.

1. **Individual verifiability:** In our model of election schemes, all recorded ballots are posted on the bulletin board. So for a voter to verify that their ballot has been recorded, it suffices to enable them to uniquely identify their ballot on the bulletin board.

   Individual verifiability experiment \( \text{Exp-IV-Ext}(\Pi, \mathcal{A}, k) \), where \( \Pi \) denotes an election scheme, \( \mathcal{A} \) denotes the adversary, and \( k \) denotes a security parameter, therefore challenges \( \mathcal{A} \) to generate a scenario in which the voter cannot uniquely identify their ballot. In essence, \( \text{Exp-IV-Ext} \) challenges \( \mathcal{A} \) to generate a collision from \text{Vote}.

   8

   If \( \mathcal{A} \) cannot win, then voters can uniquely identify their ballots on the bulletin board:

   \[
   \text{Exp-IV-Ext}(\Pi, \mathcal{A}, k) =
   \begin{align*}
   1 \quad & (PK_\tau, nC, \beta, \beta') \leftarrow \mathcal{A}(k); \\
   2 \quad & b \leftarrow \text{Vote}(PK_\tau, nC, \beta, k); \\
   3 \quad & b' \leftarrow \text{Vote}(PK_\tau, nC, \beta', k); \\
   4 \quad & \text{if } b = b' \land b \neq \perp \land b' \neq \perp \text{ then return } 1 \\
   5 \quad & \text{else return } 0 \\
   \end{align*}
   \]

   Line 1 asks \( \mathcal{A} \) to compute two candidate choices \( \beta \) and \( \beta' \), such that ballots \( b \) and \( b' \) for those choices, as computed by Vote in lines 2 and 3, are equal. Individual verifiability thus resembles Injectivity, but individual verifiability allows choices to be equal and allows \( \mathcal{A} \) to choose election parameters.

   One way to achieve individual verifiability is to base the election scheme on a probabilistic encryption scheme, such as El Gamal [41]. Intuitively, if Vote encrypts the choice using random coins, then it is overwhelmingly unlikely that two votes will result in the same ballot. Our proofs that Helios and JCJ satisfy individual verifiability are based on this idea.

   **Clash attacks:** In a clash attack [71], the adversary convinces some voters that a single ballot belongs to them all. Some clash attacks are possible because of vulnerabilities in the design of Vote. For example, if Vote simply outputs candidate choice \( \beta \), then a voter has no way to distinguish their vote for \( \beta \) from another voter’s vote for \( \beta \). Exp-IV-Ext detects clash attacks resulting from vulnerabilities in Vote.

   Some clash attacks, however, are possible because the adversary subverts the implementation of Vote. For example, the adversary might replace some hardware or software, or compromise the random number generator. If any one of these aspects is compromised, then Vote has effectively been changed to a different algorithm Vote'. The conclusions drawn by a security analyst who uses our definition of individual verifiability to analyze Vote would not necessarily be applicable to Vote'.

   In short, a voter can verify that their ballot has been recorded if and only if they run the correct Vote algorithm. We make no guarantees to voters that do not run the correct Vote algorithm. One way to make stronger guarantees is to use cut-and-choose protocols to audit ballots [11], [12]. This would require modeling voting as an interactive protocol with the adversary, rather than as an algorithm. We leave this extension as future work.

2. **Universal verifiability:** For an election to be universally verifiable, anyone must be able to check that a tally is correct with respect to recorded ballots—that is, the tally represents...
the choices used to construct the recorded ballots. Because anyone can execute Verify, it suffices that Verify accepts only when that property holds.

Universal verifiability experiment Exp-UV-Ext(\(\Pi, A, k\)) therefore challenges adversary \(A\) to concoct a scenario in which Verify incorrectly accepts:

\[
\text{Exp-UV-Ext}(\Pi, A, k) =
\begin{align*}
1 & (PK_T, BB, n_C, X, P) \leftarrow A(k); \\
2 & Y \leftarrow \text{correct-tally}(PK_T, BB, n_C, k); \\
3 & \text{if } X \neq Y \land \text{Verify}(PK_T, BB, n_C, X, P, k) = 1 \text{ then return } 1 \\
4 & \text{else return } 0
\end{align*}
\]

In line 1, \(A\) is challenged to create a bulletin board \(BB\) and purported tally \(X\) of that bulletin board. Line 2 constructs the correct tally \(Y\) of \(BB\), and line 3 checks whether Verify accepts an incorrect tally. If \(A\) cannot win Exp-UV-Ext, then Verify will not accept incorrect tallies. In particular, no ballots can be omitted from the tally, and at most one candidate choice can be included in the tally for each ballot.

Let function \(\text{correct-tally}\) be defined such that for all \(PK_T, BB, n_C, k, \ell, \) and \(\beta \in \{1, \ldots, n_C\},\)

\[
\text{correct-tally}(PK_T, BB, n_C, k)[\beta] = \ell
\iff \exists b \in (BB \setminus \{\perp\}) : \\
\exists r : b = \text{Vote}(PK_T, n_C, \beta, k; r).
\]

The vector produced by \(\text{correct-tally}\) must be of length \(n_C\). Component \(\beta\) of vector \(\text{correct-tally}(PK_T, BB, n_C, k)\) equals \(\ell\) iff there exist at least \(\ell\) ballots on the bulletin board that are votes for candidate \(\beta\). It follows that the output of \(\text{correct-tally}\) represents the choices used to construct the recorded ballots. Note that, without Injectivity, the existential quantification in \(\text{correct-tally}\) could permit a ballot to be tallied for more than one candidate. Of course, \(\text{correct-tally}\) cannot be computed by a PPT algorithm for typical cryptographic election schemes. But that does not matter, because \(\text{correct-tally}\) is never actually computed as part of an election scheme—it uses only the definition of Exp-UV-Ext.\(^{10}\)

Security analysts must convince themselves that \(\text{correct-tally}\) is indeed correct. Because of the function’s simplicity, this should be relatively straightforward. By comparison, Tally algorithms for real voting schemes tend to be complicated. For example, compare the complexity of \(\text{correct-tally}\) to Helios’s Tally algorithm, which appears in the companion technical report \(^1\).

By design, Exp-UV-Ext assumes that the ballots on bulletin board \(BB\) are exactly the ballots that should be tallied. The external authentication mechanism is assumed to prohibit unauthorized ballots from being posted on \(BB\). Helios makes such an assumption about its external authentication mechanism.

3) Eligibility verifiability: For an election to satisfy eligibility verifiability, anyone must be able to check that every tallied vote was cast by an authorized voter—that is, it must be possible to authenticate ballots. In election schemes with external authentication, a trusted third party authenticates ballots. That third party might convince itself that all tallied ballots have been authenticated, but it cannot convince all other parties. Eligibility verifiability, therefore, is not achievable in election schemes with external authentication.

4) Election verifiability: With Exp-IV-Ext and Exp-UV-Ext, we define election verifiability with external authentication. Let a PPT adversary’s success \(\text{Succ(Exp)}\) in an experiment \(\text{Exp}()\) be the probability that the adversary wins—that is, \(\text{Succ(Exp)} = \Pr[\text{Exp}()] = 1\).

Definition 4 (Ver-Ext). An election scheme \(\Pi\) satisfies election verifiability with external authentication (Ver-Ext) if for all PPT adversaries \(A\), there exists a negligible function \(\mu\), such that for all security parameters \(k\), it holds that

\[
\text{Succ(Exp-IV-Ext}(\Pi, A, k)) + \text{Succ(Exp-UV-Ext}(\Pi, A, k)) \leq \mu(k).
\]

An election scheme satisfies individual verifiability if \(\text{Succ(Exp-IV-Ext}(\Pi, A, k)) \leq \mu(k)\), and similarly for universal verifiability.

C. Example—Toy scheme from nonces

A toy election scheme satisfying Ver-Ext can be based on nonces. Each voter publishes a nonce paired with her choice to the bulletin board. This scheme illustrates the essence of election verifiability, even though it does not offer any privacy.

Definition 5. Election scheme Nonce is defined as follows:

- Setup(\(k\)) outputs (\(\perp, \perp, p_1(k), p_2(k)\)), where \(p_1\) and \(p_2\) may be any polynomial functions.
- Vote(\(PK_T, n_C, \beta, k\)) selects a nonce \(r\) uniformly at random from \(\mathbb{Z}_{2^k}\) and outputs (\(r, \beta\)).
- Tally(\(PK_T, BB, n_C, k\)) computes a vector \(X\) of length \(n_C\), such that \(X\) is a tally of the votes on \(BB\) for which the nonce is in \(\mathbb{Z}_{2^k}\), and outputs (\(X, \perp\)).
- Verify(\(PK_T, BB, n_C, X, P, k\)) outputs 1 if \((X, P) = \text{Tally}(\perp, BB, n_C, k)\) and 0 otherwise.

Proposition 1. Nonce satisfies Ver-Ext.

Proof sketch. Nonce satisfies individual verifiability, because voters can use their nonce to check that their own ballot appears on the bulletin board. With overwhelming probability, Vote will select unique nonces for each voter, hence generate distinct ballots. Nonce also satisfies universal verifiability, because plaintext candidate choices are posted on the bulletin board.
D. Orthogonality

Exp-IV-Ext and Exp-UV-Ext capture orthogonal security properties. A scheme that satisfies individual verifiability but violates universal verifiability can be constructed from Nonce by modifying Verify to always output 1. Voters can still check that their own ballot appears. But an adversary can easily win Exp-UV-Ext, because Verify will accept any tally. A scheme that satisfies universal verifiability but violates individual verifiability can be constructed from Nonce by removing the nonces, leaving just the voter’s choice in the ballots. Call that scheme Choice. Anyone can still verify the tally of the election, but an adversary can easily win Exp-IV-Ext, because two votes for the same candidate will collide.

III. Case Study: Helios

Helios is an open-source, web-based electronic voting system. Helios has been deployed in the real-world: the International Association of Cryptologic Research (IACR) has used Helios annually since 2010 to elect board members [14], [47], [53], the Catholic University of Louvain used Helios to elect the university president [6], and Princeton University has used Helios to elect several student governments [4], [74]. Attacks have been discovered against the original Helios scheme, and defenses against those attacks have been proposed [17], [18], [33], [34]. For clarity, we write Helios 2.0 to refer to the Helios scheme as originally proposed [6] and Helios 4.0 to refer to a version of Helios that incorporates the defenses [3]. When referring in general to both of these schemes, we simply write Helios.

To achieve verifiability while maintaining ballot secrecy [16], [18], Helios homomorphically encrypts candidate choices. During tallying, all encrypted choices are homomorphically combined [14] into a single ciphertext, which is then decrypted to reveal the tally. Informally, Helios works as follows:

- **Setup.** The taller generates a key pair for a homomorphic encryption scheme and publishes the public key.
- **Voting.** A voter encrypts her candidate choice with the taller’s public key, and she proves in zero knowledge that the ciphertext contains a well-formed choice. The voter posts her ballot (i.e., ciphertext and proof) on the bulletin board. During posting, the bulletin board is assumed to correctly authenticate voters.
- **Tallying.** The taller discards any ballots from the bulletin board for which proofs do not hold. The taller homomorphically combines the ciphertexts in the remaining ballots, decrypts the homomorphic combination, and proves in zero knowledge that decryption was performed correctly. Finally, the taller publishes the winning candidate and proof of correct decryption.

- **Verification.** A verifier recomputes the homomorphic combination and checks all the zero-knowledge proofs.

We give a formal description of Helios 4.0 in the companion technical report [1]. Using that formalization, we can prove that Helios 4.0 is verifiable:

**Theorem 2.** Helios 4.0 satisfies Ver-Ext.

**Proof sketch.** Helios 4.0 satisfies individual verifiability, because the probabilistic encryption scheme ensures that ballots are unique, with overwhelming probability. And Helios 4.0 satisfies universal verifiability, because the zero-knowledge proofs can be publicly verified.

A formal proof of Theorem 2 appears in the companion technical report [1]. The proof assumes the random oracle model [9].

We would not expect Ver-Ext to hold for Helios 2.0, because of known attacks [18]. Accordingly, we prove that Helios 2.0 does not satisfy Ver-Ext in the companion technical report [1].

IV. Internal Authentication

Some election schemes implement their own authentication mechanisms. JCJ [57]–[59] and Civitas [29], for example, authenticate ballots based on credentials issued to voters by a registration authority. Schemes with this kind of internal authentication enable verification of whether tallied ballots were cast by authorized voters.

A. Election scheme

A registrar is responsible for issuing authentication credentials to voters. Each voter is associated with a credential pair $(pk, sk)$. The voter uses private credential $sk$ to construct a ballot. Public credential $pk$ is used during tallying and verification. Let $L$ denote the electoral roll, which is the set of all public credentials.

An election scheme with internal authentication, which henceforth in this section we abbreviate as “election scheme,” is a tuple $(\text{Setup, Register, Vote, Tally, Verify})$ of PPT algorithms. The algorithms are now denoted as follows:

- $(PK_T, SK_T, m_B, m_C) \leftarrow \text{Setup}(k)$
- $(pk, sk) \leftarrow \text{Register}(PK_T, k)$
- $b \leftarrow \text{Vote}(sk, PK_T, n_C, \beta, k)$
- $(X, P) \leftarrow \text{Tally}(PK_T, SK_T, BB, L, n_C, k)$
- $v \leftarrow \text{Verify}(PK_T, BB, L, n_C, X, P, k)$

Setup is unchanged from election schemes with external authentication (cf. III-A). The only change to Vote is that it now accepts private credential $sk$ as input. Similarly, the only change to Tally and Verify is that they now accept electoral credentials [16]. Some election schemes (e.g., JCJ) permit the registrar’s role to be distributed among several registrars. For simplicity, we consider only a single registrar in this paper.

---

12https://vote.heliosvoting.org/
13Our analysis of Helios 4.0 is based on the specification [5] for the next release. This specification incorporates proposals by Cortier and Smyth [34] for non-malleable ballots and by Bernhard et al. [18] to replace the weak Fiat–Shamir transformation with the strong Fiat–Shamir transformation.
14The homomorphic combination of ciphertexts is straightforward for two-candidate elections [10], [15], [30], [31], [37], since choices (e.g., “yes” or “no”) can be encoded as 1 or 0. Multi-candidate elections are also possible [15], [30], [37].
15Our formalization is the first cryptographic description of Helios 4.0, hence an additional contribution of this work.
16Some election schemes (e.g., JCJ) permit the registrar’s role to be distributed among several registrars. For simplicity, we consider only a single registrar in this paper.
roll \( L \) as input. Register is executed by the registrar. It takes as input the public key \( PK_T \) of the teller and security parameter \( k \), and it outputs a credential pair \( (pk, sk) \). After all voters have been registered, the registrar certifies the electoral roll, perhaps by digitally signing and publishing it.\footnote{It might seem surprising that Register does not require the registrar to provide any private keys as input. But in constructions of election schemes with internal authentication, e.g., \cite{29,59}, the registrar does not sign credential pairs with its own private key. Rather, the registrar signs the electoral roll.}

Election schemes must continue to satisfy Correctness, Completeness, and Injectivity, which we update to include private credentials and the electoral roll:

**Definition 6** (Correctness). There exists a negligible function \( \mu \), such that for all security parameters \( k \), integers \( n_B \) and \( n_C \), and choices \( \beta_1, \ldots, \beta_{n_B} \in \{1, \ldots, n_C \} \), it holds that if \( Y \) is a vector of length \( n_C \) whose components are all 0, then

\[
\Pr[(PK_T, SK_T, m_B, m_C) \leftarrow \text{Setup}(k); \\
\quad \text{for } 1 \leq i \leq n_B \text{ do} \\
\quad \quad (pk_i, sk_i) \leftarrow \text{Register}(PK_T, k); \\
\quad \quad b_i \leftarrow \text{Vote}(sk_i, PK_T, n_C, \beta_i, k); \\
\quad \quad Y[\beta_i] \leftarrow Y[\beta_i] + 1; \\
\quad L \leftarrow \{pk_1, \ldots, pk_{n_B}\}; \\
\quad BB \leftarrow \{b_1, \ldots, b_{n_B}\}; \\
\quad (X, P) \leftarrow \text{Tally}(PK_T, SK_T, BB, L, n_C, k): \\
\quad n_B \leq m_B \land n_C \leq m_C \Rightarrow X = Y > 1 - \mu(k).]
\]

**Definition 7** (Completeness). There exists a negligible function \( \mu \), such that for all security parameters \( k \), bulletin boards \( BB \), and integers \( n_C \) and \( n_V \), it holds that

\[
\Pr[(PK_T, SK_T, m_B, m_C) \leftarrow \text{Setup}(k); \\
\quad \text{for } 1 \leq i \leq n_V \text{ do} (pk_i, sk_i) \leftarrow \text{Register}(PK_T, k); \\
\quad L \leftarrow \{pk_1, \ldots, pk_{n_V}\}; \\
\quad (X, P) \leftarrow \text{Tally}(PK_T, SK_T, BB, L, n_C, k): \\
\quad \text{Verify}(PK_T, BB, L, n_C, X, P, k) = 1] > 1 - \mu(k).]
\]

**Definition 8** (Injectivity). For all security parameters \( k \), public keys \( PK_T \), integers \( n_C \), and choices \( \beta \) and \( \beta' \), such that \( \beta \neq \beta' \), we have

\[
\Pr[(pk, sk) \leftarrow \text{Register}(PK_T, k); \\
\quad (pk', sk') \leftarrow \text{Register}(PK_T, k); \\
\quad b \leftarrow \text{Vote}(sk, PK_T, n_C, \beta, k); \\
\quad b' \leftarrow \text{Vote}(sk', PK_T, n_C, \beta', k): \\
\quad b \neq \perp \land b' \neq \perp \Rightarrow b = b'.]
\]

### B. Election verifiability

Recall (from \cite{II-B}) that election verifiability is expressed with experiments, and that an adversary wins by causing an experiment to output 1. We henceforth assume that the adversary is stateful—that is, information persists across invocations of the adversary in a single experiment. Our experiments in Section \cite{II} did not need this assumption, because they never invoked the adversary more than once.

In our experiments, below, we model an adversary who cannot corrupt the registration process that issues credentials to voters.\footnote{Küsters and Truderung \cite{65} explore some consequences of permitting adversarial influence during registration.} Hence our definitions will not detect attacks against verifiability that result solely from weaknesses in the registration process. Secure construction of electoral rolls is not a topic that electronic voting usually addresses—though it seems an important part of any real-world deployment.

1) **Individual verifiability:** The individual verifiability experiment again challenges adversary \( A \) to generate a scenario in which the voter could not uniquely identify their ballot.\footnote{Unlike Exp-IV-Ext, a variant of Exp-IV-Int that challenges \( A \) to predict the output of Vote is strictly stronger. See the companion technical report \cite{1} for details.}

\[
\text{Exp-IV-Int}(\Pi, A, k) = \\
1 \quad (PK_T, n_V) \leftarrow A(k); \\
2 \quad \text{for } 1 \leq i \leq n_V \text{ do} (pk_i, sk_i) \leftarrow \text{Register}(PK_T, k) \\
3 \quad L \leftarrow \{pk_1, \ldots, pk_{n_V}\}; \\
4 \quad \text{Crpt} \leftarrow \emptyset; \\
5 \quad (n_C, \beta, \beta', i, j) \leftarrow \text{A}^*(L); \\
6 \quad b \leftarrow \text{Vote}(sk_i, PK_T, n_C, \beta, k); \\
7 \quad b' \leftarrow \text{Vote}(sk_j, PK_T, n_C, \beta', k); \\
8 \text{if } b = b' \neq \perp \land b' \neq \perp \land i \neq j \land sk_i \notin \text{Crpt} \land sk_j \notin \text{Crpt} \text{ then } \\
9 \quad \quad \text{return } 1 \\
10 \text{else } \\
11 \quad \quad \text{return } 0
\]

The main differences from the corresponding experiment for external authentication (\cite{II-B}) are that voters are registered in line 2, and that \( A \) is given access to an oracle \( C \) in line 5. The oracle is used to model \( A \) corrupting voters and learning their private credentials: on invocation \( C(\ell) \), where \( 1 \leq \ell \leq n_V \), the oracle records that voter \( \ell \) is corrupted by updating \( \text{Crpt} \) to be \( \text{Crpt} \cup \{sk_\ell\} \) and outputs \( sk_\ell \). In line 5, the voter indices output by \( A \) must be legal with respect to \( n_V \), but we elide that detail from the experiment for simplicity. Line 8 ensures that \( A \) cannot trivially win by corrupting voters.

2) **Universal verifiability:** The universal verifiability experiment again challenges \( A \) to concoct a scenario in which \( \text{Verify} \) incorrectly accepts:
Exp-UV-Int(\(\Pi, A, k\)) =
\[
1 \ (PK_T, nV) \leftarrow A(k);
\]
\[
2 \ for \ 1 \leq i \leq nV \ do \ (pk_i, sk_i) \leftarrow \text{Register}(PK_T, k);
\]
\[
3 \ L \leftarrow \{pk_1, \ldots, pk_{nV}\};
\]
\[
4 \ M \leftarrow \{(pk_1, sk_1), \ldots, (pk_{nV}, sk_{nV})\};
\]
\[
5 \ (BB, nC, X, P) \leftarrow A(M);
\]
\[
6 \ Y \leftarrow \text{correct-tally}(PK_T, BB, M, nC, k);
\]
\[
7 \ if \ X \neq Y \land \text{Verify}(PK_T, BB, L, nC, X, P, k) = 1 \ then
\]
\[
8 \ \ return \ 1
\]
\[
9 \ else
\]
\[
10 \ \ return \ 0
\]

The main differences from the corresponding experiment for external authentication (cf. \([\text{II-B2}]\)) are that voters are registered in line 2, and their credential pairs are used in the rest of the experiment.

Function \(\text{correct-tally}\) is now modified to tally only authorized ballots. A ballot is \(\text{authorized}\) if it is constructed with a private credential from \(M\), and that private credential was not used to construct any other ballot on \(BB\). By comparison, the original \(\text{correct-tally}\) function (cf. \([\text{II-B2}]\)) tallies all the ballots on \(BB\).

Formally, let function \(\text{correct-tally}\) now be defined such that for all \(PK_T, BB, M, nC, k, \ell, \beta \in \{1, \ldots, nC\}\),

\[
\text{correct-tally}(PK_T, BB, M, nC, k)[\beta] = \ell
\]

\(\iff\) \(\exists b \in \text{authorized}(PK_T, (BB \setminus \{\bot\}), M, nC, k) : \exists sk, r : b = \text{Vote}(sk, PK_T, nC, \beta, k; r).\)

Let \(\text{authorized}\) be defined as follows:

\[
\text{authorized}(PK_T, BB, M, nC, k) =
\]

\[
\{b : b \in BB \land \exists pk, sk, \beta, r : b = \text{Vote}(sk, PK_T, nC, \beta, k; r) \land (pk, sk) \in M \land \neg \exists b', \beta', r' : b' \in (BB \setminus \{b\}) \land b' = \text{Vote}(sk, PK_T, nC, \beta', k; r').\}
\]

Function \(\text{authorized}\) discards all revotes—that is, if there is more than one ballot submitted with a private credential \(sk\), then all ballots submitted under that credential are discarded. Therefore, election schemes that permit revoting cannot be analyzed by this definition of \(\text{authorized}\). But alternative definitions of \(\text{authorized}\) are possible—for example, if ballots were timestamped, \(\text{authorized}\) could discard all but the most recent ballot submitted under a particular credential.

3) Eligibility verifiability: Recall (from \([\text{II-B3}]\)) that for an election scheme to satisfy eligibility verifiability, anyone must be able to check that every tallied vote was cast by an authorized voter—that is, it must be possible to authenticate ballots. Because voters are issued credential pairs that can be used to authenticate ballots, it suffices to ensure that knowledge of a private credential is necessary to construct an authentic ballot.

Eligibility verifiability experiment \(\text{Exp-EV-Int}\) therefore challenges \(A\) to produce a ballot under a private credential that \(A\) does not know:

Exp-EV-Int(\(\Pi, A, k\)) =
\[
1 \ (PK_T, nV) \leftarrow A(k);
\]
\[
2 \ for \ 1 \leq i \leq nV \ do \ (pk_i, sk_i) \leftarrow \text{Register}(PK_T, k);
\]
\[
3 \ L \leftarrow \{pk_1, \ldots, pk_{nV}\};
\]
\[
4 \ \text{Crpt} \leftarrow \emptyset; \ \text{Rvld} \leftarrow \emptyset;
\]
\[
5 \ (nC, \beta, i, b) \leftarrow A.C.R(L);
\]
\[
6 \ if \ \exists r : b = \text{Vote}(sk_i, PK_T, nC, \beta, k; r) \land b \neq \bot \land b \notin \text{Rvld} \land sk_i \notin \text{Crpt} \ then
\]
\[
7 \ \ return \ 1
\]
\[
8 \ else
\]
\[
9 \ \ return \ 0
\]

In line 1, \(A\) chooses the teller’s public key and the number of voters. Line 2 registers voters. \(A\) is not permitted to influence registration while it is in progress. In particular, \(A\) is not permitted to choose credential pairs, because by doing so \(A\) could trivially win the experiment.

Line 4 initializes two sets: \(\text{Crpt}\) is a set of voters who have been corrupted, meaning that \(A\) has learned their private credential, and \(\text{Rvld}\) is a set of ballots that have been revealed to \(A\). The former set models \(A\) coercing voters to reveal their private credentials. The latter set models \(A\) observing ballots on the bulletin board.

Line 5 challenges \(A\) to produce a ballot \(b\) with the help of two oracles. Oracle \(C\) is the same oracle as in \(\text{Exp-IV-Int}\) (cf. \([\text{IV-B1}]\)); it leaks the private credentials of corrupted voters to \(A\). Oracle \(R\) reveals ballots. On invocation \(R(i, \beta, nC)\), where \(1 \leq i \leq nV\), oracle \(R\) does the following:

- Computes a ballot \(b\) that represents a vote for candidate \(\beta\) by a voter with private credential \(sk_i\), that is, computes \(b \leftarrow \text{Vote}(sk_i, PK_T, nC, \beta, k)\).
- Records \(b\) as being revealed by updating \(\text{Rvld}\) to be \(\text{Rvld} \cup \{b\}\).
- Outputs \(b\).

In line 6, \(A\) wins if (i) the ballot is \(\text{authentic}\), meaning that it is the output of \(\text{Vote}\) on an authorized credential, and (ii) that credential belongs to a voter that \(A\) did not corrupt, and (iii) that ballot was not revealed. If \(A\) cannot succeed in this experiment, then only authorized votes are tallied.

4) Election verifiability: With \(\text{Exp-IV-Int}, \text{Exp-UV-Int}, \text{and Exp-EV-Int}\), we define election verifiability with internal authentication.

**Definition 9 (Ver-Int).** An election scheme \(\Pi\) satisfies election verifiability with internal authentication \((\text{Ver-Int})\) if for all PPT adversaries \(A\), there exists a negligible function \(\mu\), such that for all security parameters \(k\), it holds that

\[\text{Succ}(\text{Exp-IV-Int}(\Pi, A, k)) + \text{Succ}(\text{Exp-UV-Int}(\Pi, A, k)) + \text{Succ}(\text{Exp-EV-Int}(\Pi, A, k)) \leq \mu(k).\]

An election scheme satisfies eligibility verifiability if \(\text{Succ}(\text{Exp-EV-Int}(\Pi, A, k)) \leq \mu(k)\), and similarly for individual and universal verifiability.

C. Example—Toy schemes from digital signatures

A toy election scheme satisfying Ver-Int can be based on a digital signature scheme \((\text{Gen}, \text{Sign}, \text{Ver})\). Each voter
presents their signed candidate choice on the bulletin board.

**Definition 10.** Election scheme $\text{Sig}$ is defined as follows:

- Setup($k$) outputs $(\bot, \bot, p_1(k), p_2(k))$, where $p_1$ and $p_2$ may be any polynomial functions.
- Register($PK_{\tau}, k$) computes $(pk, sk) \leftarrow \text{Gen}(1^k)$ and outputs $(pk, sk)$.
- Vote($sk, PK_{\tau}, n_C, \beta, k$) outputs $(\beta, \text{Sign}(sk, \beta))$.
- Tally($PK_{\tau}, SK_{\tau}, BB, L, n_C, k$) computes a vector $X$ of length $n_C$, such that $X$ is a tally of all the ballots on $BB$ that are signed by distinct private keys whose corresponding public keys appear in $L$, and outputs $(X, \bot)$.
- Verify($PK_{\tau}, BB, L, n_C, X, P, k$) outputs 1 if $(X, P) = \text{Tally}(\bot, \bot, BB, L, n_C, \bot)$ and 0 otherwise.

The verifiability of $\text{Sig}$ follows from the security of the underlying signature scheme:

**Proposition 3.** If $(\text{Gen, Sign, Ver})$ is a signature scheme satisfying existential unforgeability under adaptive chosen-message attack, then $\text{Sig}$ satisfies $\text{Ver-Int}$.

**Proof sketch.** $\text{Sig}$ satisfies individual verifiability, because voters can verify that their signed choices appear on the bulletin board. $\text{Sig}$ satisfies universal verifiability, because signed plaintext choices are posted on $BB$. Finally, $\text{Sig}$ satisfies eligibility verifiability, because anyone can check that the signed choices belong to registered voters.

**D. Orthogonality**

Exp-EV-Int, Exp-UV-Int, and Exp-EV-Int capture mostly orthogonal security properties, as shown in Table I. Individual and universal verifiability are orthogonal, and eligibility verifiability implies individual verifiability.

**Theorem 4.** If an election scheme $\Pi$ satisfies Exp-EV-Int, then $\Pi$ also satisfies Exp-EV-Int.

**Proof sketch.** If $\Pi$ satisfies Exp-EV-Int, then no one can construct a ballot that appears to be associated with public credential $pk$ unless they know private credential $sk$. That means that a voter can uniquely identify their ballot, because no one else knows their private credential. Therefore $\Pi$ satisfies Exp-EV-Int.

The proof of Theorem 4 appears in the companion technical report [1].

In Table I, AlwaysVerify($\cdot$) is a function that transforms an election scheme by compromising Verify to always return 1. Thus, AlwaysVerify($\Pi$) is guaranteed not to satisfy Exp-UV-Int. Similarly, IgnoreCreds($\cdot$) is a function that accepts as input an election scheme with external authentication and returns as output an election scheme with internal authentication. The resulting scheme, however, simply ignores credentials altogether: Register returns $(\bot, \bot)$, Vote ignores $sk$, and Tally and Verify ignore $L$. Thus, IgnoreCreds($\Pi$) is guaranteed not to satisfy Exp-EV-Int. Using those functions, we briefly explain each line of the table:

<table>
<thead>
<tr>
<th>Line</th>
<th>IV</th>
<th>UV</th>
<th>EV</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>AlwaysVerify(IgnoreCreds(Choice))</td>
</tr>
<tr>
<td>2</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>IgnoreCreds(Choice)</td>
</tr>
<tr>
<td>3</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>AlwaysVerify(IgnoreCreds(Nonce))</td>
</tr>
<tr>
<td>4</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>AlwaysVerify(Sign)</td>
</tr>
<tr>
<td>5</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>Malleable Sig</td>
</tr>
<tr>
<td>6</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>Sig</td>
</tr>
</tbody>
</table>

**TABLE I**

**ELECTION SCHEMES THAT SATISFY EACH COMBINATION OF INDIVIDUAL, UNIVERSAL AND ELIGIBILITY VERIFIABILITY**

1) Recall (from [21]) that Choice is the election scheme in which ballots contain only the plaintext candidate choice. By compromising Verify and ignoring credentials, we obtain a scheme that satisfies no properties.
2) By Theorem 4 this situation is impossible.
3) Compared to line 1 of Table I this scheme satisfies Exp-UV-Int, because Verify is not compromised.
4) By Theorem 4 this situation is impossible.
6) $\text{Sig}$ satisfies all three properties. By compromising Verify, we obtain a scheme that satisfies only Exp-EV-Int and Exp-EV-Int.
7) By making $\text{Sig}$’s underlying signature scheme malleable, we could obtain a scheme that does not satisfy $\text{Exp-EV-Int}$, because the adversary could construct a valid ballot out of a revealed ballot. But the scheme would continue to satisfy Exp-EV-Int and Exp-UV-Int.
8) $\text{Sig}$ satisfies all three properties.

**V. CASE STUDY: JCJ**

JCJ (named for its designers, Juels, Catalano, and Jakobsen) [37], [59] is a coercion-resistant election scheme, meaning voters cannot prove whether or how they voted, even if they can interact with the adversary while voting. Coercion resistance protects elections from improper influence by adversaries.

To achieve verifiability and coercion resistance, JCJ uses verifiable mixnets, which anonymize a set of messages [3]. During tallying, all encrypted choices are anonymized by a mixnet, then all choices are decrypted. The tally is computed from the decrypted choices. Informally, JCJ works as follows:

- **Setup.** The taller generates a key pair $(PK_{\tau}, SK_{\tau})$ for an encryption scheme and publishes the public key.
- **Registration.** To register a voter, the registrar generates a nonce, which is sent to the voter and serves as the private credential. The public credential is computed as an

---

20) Given a message $m$ and signature $\sigma$, a malleable signature scheme permits computation of a signature $\sigma'$ on a related message $m'$ [21]. The malleable signature scheme $\text{Sig}$ used in line 7 of Table I would need to enable an adversary to transform a signature on a well-formed candidate $\beta$ into a signature on a distinct, well-formed candidate $\beta'$.
encryption of the private credential with $PK_T$. After all voters are registered, the registrar publishes the electoral roll.

- **Voting.** A voter encrypts her candidate choice with $PK_T$. She also encrypts her private credential with $PK_T$. She proves in zero-knowledge that she simultaneously knows both plaintexts, and that her choice is well-formed. The voter posts her ballot (i.e., both ciphertexts and the proof) on the bulletin board.

- **Tallying.** The tallier discards any ballots from the bulletin board for which the zero-knowledge proofs do not verify. All unauthorized ballots are then discarded through a combination of protocols that includes verifiable mixnets and plaintext equivalence tests (PETs) [54]. (PETs enable proof that two ciphertexts contain the same plaintext without revealing that plaintext.) The tallier decrypts and publishes the remaining ballots, along with a proof that decryption was performed correctly.

- **Verification.** A verifier checks all the proofs included in ballots, and all the proofs published during tallying.

The companion technical report [1] gives a formal description of JCJ. That formalization satisfies individual and universal verifiability, assuming that the cryptographic primitives satisfy certain properties that we identify. But the formalization fails to satisfy eligibility verifiability, because knowledge of the tallier’s private key $SK_T$ suffices to construct ballots that appear authentic: with $SK_T$, any public credential can be decrypted to discover the corresponding private credential. Note that Exp-EV-Int allows an adversary $\mathcal{A}$ to choose the tallier’s key pair, so $\mathcal{A}$ does know $SK_T$ hence can construct a ballot that suffices to win Exp-EV-Int.

We can nonetheless prove that JCJ satisfies a variant of eligibility verifiability. Consider the following experiment, which does not permit the adversary to choose the tallier’s key pair:

$$\text{Exp-EV-Int-Weak}(\Pi, \mathcal{A}, k) =$$

1. $(PK_T, SK_T, n_B, n_C) \leftarrow \text{Setup}(k)$;
2. $n_V \leftarrow \mathcal{A}(PK_T, k)$;
3. for $1 \leq i \leq n_V$ do $(pk_i, sk_i) \leftarrow \text{Register}(PK_T, k)$;
4. $L \leftarrow \{pk_1, \ldots, pk_{n_V}\}$;
5. $\text{Crpt} \leftarrow \emptyset$; $\text{Rold} \leftarrow \emptyset$;
6. $(n_C, \beta, i, b) \leftarrow A^{C,R}(L)$;
7. if $\exists r : b = \text{Vote}(sk_i, PK_T, n_C, \beta, k; r) \land b \neq \bot$ then $\text{Rold} \land sk_i \notin \text{Crpt}$ then
8. return 1
9. else
10. return 0

Line 1 of Exp-EV-Int has been refactored into lines 1 and 2 of Exp-EV-Int-Weak. In line 1 of Exp-EV-Int-Weak, keys are generated by the experiment. In line 2, $\mathcal{A}$ is given the public key but not the private key.

Using Exp-EV-Int-Weak, we define a weaker variant of Ver-Int and prove that JCJ satisfies it:

**Definition 11 (Ver-Int-Weak).** An election scheme $\Pi$ satisfies weak election verifiability with internal authentication (Ver-Int-Weak) if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function $\mu$, such that for all security parameters $k$, we have $\text{Succ}(\text{Exp-IV-Int}(\Pi, \mathcal{A}, k)) + \text{Succ}(\text{Exp-UV-Int}(\Pi, \mathcal{A}, k)) + \text{Succ}(\text{Exp-EV-Int-Weak}(\Pi, \mathcal{A}, k)) \leq \mu(k)$.

**Theorem 5.** JCJ satisfies Ver-Int-Weak.

**Proof sketch.** JCJ satisfies individual verifiability, because the probabilistic encryption scheme ensures that ballots are unique, with overwhelming probability. JCJ satisfies universal verifiability, because the proofs produced throughout tallying can be publicly verified. And JCJ satisfies eligibility verifiability, because $\mathcal{A}$ cannot construct new ballots without knowing a voter’s private credential or the tallier’s private key.

A formal proof of Theorem 5 appears in the companion technical report [1]. The proof assumes the random oracle model.

The Civitas [29] scheme refines the JCJ scheme. Some refinements relevant to election verifiability are an implementation of a distributed registration protocol, and a mixnet based on randomized partial checking (RPC) [55]. We leave a proof that Civitas satisfies Ver-Int-Weak as future work.

**VI. NEW CLASSES OF ATTACK**

Our definitions of election verifiability improve upon existing definitions by detecting two previously unidentified classes of attack:

- **Collusion attacks.** An election scheme’s tallying and verification algorithms might be designed such that they collude to accept incorrect tallies.

- **Biasing attacks.** An election scheme’s verification algorithm might be designed such that it rejects some legitimate tallies.

Although a well-designed election scheme would hopefully not exhibit these vulnerabilities, it is the job of verifiability definitions to detect malicious schemes, regardless of whether vulnerabilities are due to malice or errors. So definitions of election verifiability should preclude collusion and biasing attacks.

**A. Collusion Attacks**

Here are two examples of potential collusion attacks:

- **Vote stuffing.** Tally behaves normally, but adds $\kappa$ votes for candidate $\beta$. Verify subtracts $\kappa$ votes from $\beta$, then proceeds with verification as normal. Elections thus verify as normal, except that candidate $\beta$ receives extra votes.

- **Backdoor tally replacement.** Tally and Verify behave normally, unless a backdoor value is posted on the bulletin board $BB$. For example, if $(SK_T, X^*)$ appears on $BB$, then Tally and Verify both ignore the correct tally and instead replace it with tally $X^*$. Value $SK_T$ is the backdoor here; it cannot appear on $BB$ (except with negligible probability) unless the tallier is malicious.
Vote stuffing is detected by our definitions of Correctness (§II-A and §IV-A), because these definitions require that the tally produced by Tally corresponds to the choices encapsulated in ballots on the bulletin board. Note that vote stuffing is not a failure of eligibility verifiability, because the stuffed votes do not correspond to any ballots on the bulletin board. Backdoor tally replacement is detected by our definitions of universal verifiability (§II-B2 and §IV-B2), because those definitions require Verify to accept only those tallies that correspond to a correct tally of the bulletin board.

We show, next, that the definition of election verifiability by Juels et al. [59] fails to detect vote stuffing and backdoor tally replacement, and that the definition by Cortier et al. [32] fails to detect backdoor tally replacement.

Juels et al. [59] formalize definitions that we name JCJ-correctness and JCJ-verifiability. JCJ-correctness is intuitively meant to capture that “A cannot pre-empt, alter, or cancel the votes of honest voters [and] that A cannot cause voters to cast ballots resulting in double voting” [59, p. 45]; it is formalized in terms of whether the adversary can post ballots on the bulletin board that cause the tally to be computed incorrectly. JCJ-verifiability is intuitively “the ability for any player to check whether the tally...has been correctly computed” [59, p. 46]; it is formalized in terms of whether Verify will accept a tally that differs from the output of Tally. We restate the formal definitions in the companion technical report [1].

To show that the JCJ definitions fail to detect collusion attacks, we first formalize the vote stuffing attack. An election scheme $\Pi = (\ldots, Tally, Verify)$ can be modified to derive a vote-stuffing election scheme $\text{Stuff}(\Pi, \beta, \kappa) = (\ldots, Tally_S, Verify_S)$, which adds $\kappa$ votes to $\beta$, as follows:

- $Tally_S(\ldots)$ computes $(X, P) \leftarrow Tally(\ldots)$ and outputs $(\text{Add}(X, \beta, \kappa), P)$
- $Verify_S(\ldots, X, P, \ldots)$ computes $v \leftarrow Verify(\ldots, \text{Add}(X, \beta, -\kappa), P, \ldots)$ and outputs $v$.

**Proposition 6.** Given an election scheme $\Pi$ satisfying JCJ-correctness and JCJ-verifiability, and integers $\beta$ and $\kappa$, it holds that $\text{Stuff}(\Pi, \beta, \kappa)$ satisfies JCJ-correctness and JCJ-verifiability.

A formal proof appears in the companion technical report [1].

Second, we formalize the backdoor tally replacement attack. An election scheme $\Pi = (\ldots, Tally, Verify)$ can be modified to derive an election scheme $\text{Backdoor}(\Pi) = (\ldots, Tally_B, Verify_B)$, as follows:

- $Tally_B(\ldots, SK_T, BB, \ldots)$ checks whether there exists a unique $X^*$ such that $(SK_T, X^*) \in BB$. If so, $Tally_B$ outputs $(X^*, \bot)$. Otherwise, $Tally_B$ computes $(X, P) \leftarrow Tally(\ldots, SK_T, BB, \ldots)$ and outputs $(X, P)$.
- $Verify_B(PK_T, BB, \ldots, X, P, \ldots)$ checks whether there exists a unique $X^*$ such that $(SK_T, X^*) \in BB$. If so, $Verify_B$ outputs 1. Otherwise, $Verify_B$ computes $v \leftarrow Verify(\ldots, PK_T, BB, \ldots, X, P, \ldots)$ and outputs $v$.

**Proposition 7.** Given an election scheme $\Pi$ satisfying JCJ-correctness and JCJ-verifiability that does not leak the taller’s private key, it holds that $\text{Backdoor}(\Pi)$ satisfies JCJ-correctness and JCJ-verifiability.

A formal proof appears in the companion technical report [1], where we also formally define key leakage.

Cortier et al. [32] propose definitions similar to JCJ-verifiability and insist that election schemes must satisfy their notions of correctness and partial tallying. Vote stuffing is detected by their correctness property, but backdoor tally replacement is not. The ideas remain the same, so we omit formalized results. We have reported these findings to the original authors [31], [43], [44].

### B. Biasing attacks

Here are three formalizations of biasing attacks, derived from an election scheme $\Pi = (\ldots, \text{Verify})$.

- **Reject All.** Let $\text{Reject}(\Pi) = (\ldots, \text{Verify}_R)$, where $\text{Verify}_R$ always outputs 0. $\text{Verify}_R$ therefore always rejects, hence no election can ever be considered valid.

- **Selective Reject.** Let $\varepsilon$ be a distinguished value that would not be posted on the bulletin board by honest voters. Let $\text{Selective}(\Pi, \varepsilon) = (\ldots, \text{Verify}_R)$, where $\text{Verify}_R(\ldots, BB, \ldots)$ computes $v \leftarrow \text{Verify}(\ldots, BB, \ldots)$ and outputs 1 if both $v = 1$ and $\varepsilon \notin BB$. Otherwise, $\text{Verify}_R$ outputs 0. $\text{Verify}_R$ therefore rejects if $\varepsilon$ appears on the bulletin board, hence some elections can be invalidated.

- **Biased Reject.** Suppose $Z$ is a set of tallies. Let $\text{Bias}(\Pi, Z) = (\ldots, \text{Verify}_R)$, where $\text{Verify}_R(\ldots, X, \ldots)$ computes $v \leftarrow \text{Verify}(\ldots, X, \ldots)$ and outputs 1 if both $v = 1$ and $X \in Z$. Otherwise, $\text{Verify}_R$ outputs 0. $\text{Verify}_R$ therefore only accepts a subset of the tallies accepted by Verify, hence biases tallies toward $Z$.

These formalizations do not satisfy our definition of Completeness (§II-A and §IV-A), hence, our definitions of verifiability detect these biasing attacks.

The definition of verifiability by Juels et al. [59] fails to detect all three of the above attacks, because that definition has no notion of Completeness. For example, it is vulnerable to Biased Reject attacks:

**Proposition 8.** Given an election scheme $\Pi$ satisfying JCJ-correctness and JCJ-verifiability, and given a multiset $Z$, it holds that $\text{Bias}(\Pi, Z)$ satisfies JCJ-correctness and JCJ-verifiability.

A formal proof appears in the companion technical report [1].

The definition of verifiability by Kiayias et al. [61] fails to detect Selective Reject attacks, because (like JCJ) the

22 We omit many of the parameters of Tally and Verify here for simplicity; see the companion technical report [1] for details.

23 Let $\text{Add}(X, \beta, \kappa) = (X[1], \ldots, X[\beta - 1], X[\beta] + \kappa, X[\beta + 1], \ldots, X[|X|])$. And let $|X|$ denote the length of vector $X$.

24 Verify$_B$ also needs to check that $SK_T$ is the private key corresponding to $PK_T$. We omit formalizing this detail, but note that it is straightforward for real-world encryption schemes such as El Gamal and RSA.
Our definition of election verifiability follows Smyth et al. [63, 80, 83] by deconstructing it into individual, universal, and eligibility verifiability. Other deconstructions of election verifiability are possible. For example, Adida and Neff [7] §2 identify four aspects of verifiability:

- **Cast as intended:** the ballot is cast at the polling station as the voter intended.
- **Recorded as cast:** cast ballots are preserved with integrity through the ballot collection process.
- **Counted as recorded:** recorded ballots are counted correctly.
- **Eligible voter verification:** only eligible voters can cast a ballot in the first place.

Those definitions are not mathematical, so we cannot attempt a precise comparison. Nonetheless, eligibility verifiability and eligible voter verification seem to be addressing similar concerns. Likewise, individual and universal verifiability together seem to be addressing concerns similar to that of recorded as cast and counted as recorded together. Recorded as cast, in our work, reduces to the bulletin board preserving ballots with integrity—a property that we have assumed, because cryptographic election schemes assume it too. Ways to construct secure bulletin boards have been proposed, e.g., [36], [48], [76], [78]. We postpone a discussion of cast as intended to Section VIII.

Privacy properties [38], [59], [69], [70], [81], [82]—such as ballot secrecy, receipt freeness, and coercion resistance—complement verifiability. Chevallier-Mames et al. [26], [27] and Hosp and Vora [51], [52] show an incompatibility result: election schemes cannot unconditionally satisfy privacy and universal verifiability. But weaker versions of these properties can hold simultaneously, as can be witnessed from Theorems 2 and 5 coupled with existing privacy results such as the ballot secrecy proofs for Helios 4.0 [18, Theorem 3], [16, Theorem 6.12], and the coercion resistance proof for JCI [59 §5].

**Comparison with global verifiability:** Küsters et al. [68], [69], [71] present a definition of global verifiability that can be used with any kind of protocol, not just electronic voting protocols. To analyze the verifiability of a protocol, users of this definition must themselves formalize goals, which are properties required to hold in every run of the protocol. For example, a goal $\gamma$ is presented in a case study [69 §5.2] of global verifiability applied to voting:

$\gamma$ contains all runs for which there exist choices of the dishonest voters (where a choice is either to abstain or to vote for one of the candidates) such that the result obtained together with the choices made by the honest voters in this run differs only by $\ell$ votes from the published result (i.e. the result that can be computed from the simple ballots on the bulletin board).

Another goal $\gamma$ is presented in a case study [71 §6.2] of Helios:

$\gamma$ is satisfied in a run if the published result exactly
reflects the actual votes of the honest voters in this run and votes of dishonest voters are distributed in some way on the candidates, possibly in a different way than how the dishonest voters actually voted.

These informal statements of goals are appealing, but they do not constitute rigorous mathematical definitions. As Kiayias et al. write, “[g]lobal verifiability] has the disadvantage that the set $\gamma$ remains undetermined and thus the level of verifiability that is offered by the definition hinges on the proper definition of $\gamma$ which may not be simple” [61, p. 476]. In our own work, we found that formal definitions were quite tricky to get right—for example, which ballots should be counted, how to count them, and how to determine whether that count differed from the published tally. So we shared [65] and discussed [66] our results with Küsters. In response, Küsters et al. updated an online technical report to propose a formalization of goals [64, §5.2]; we look forward to analyzing that formalization when it is published.

In an analysis of Helios, Küsters et al. [71] use goal $\gamma$ to conclude that Helios 2.0 satisfies global verifiability. Yet Bernhard et al. [18] demonstrate an attack against the verifiability of Helios 2.0, and in the companion technical report [1] we show that Helios 2.0 does not satisfy Ver-Ext. This seeming discrepancy arises because the analysis in [71] does not formalize all the cryptographic primitives used by Helios, hence the attack goes unnoticed. So another contribution of our own work is to correctly distinguish between unverifiable and verifiable variants of Helios by rigorously analyzing the cryptography used in Helios.

It is natural to ask whether election verifiability can be expressed in terms of global verifiability. We believe it can be. For instance, individual, universal and eligibility verifiability could be expressed, in the informal style of the goals quoted above, as the following goals:

- $\gamma_{IV}$ is satisfied in a run if voters can uniquely identify their ballots on the bulletin board in this run.
- $\gamma_{UV}$ is satisfied in a run if the correct tally of votes cast by authorized voters in this run is the same as the tally produced by algorithm Tally.
- $\gamma_{EV}$ is satisfied in a run if every ballot tallied in this run was created by a voter in possession of a private credential.

Küsters et al. [69] argue that deconstructing verifiability into individual and universal verifiability is insufficient to detect certain attacks involving ill-formed ballots. But those attacks leave open the possibility that there do exist notions of individual and universal verifiability that would be sufficient. Indeed, our own definition of universal verifiability rules out attacks based on ill-formed ballots, because $correct$-tally ensures that tallied ballots are well-formed.

One concern that might be raised is whether there still lurk any “gaps” in our decomposition into individual and universal (and eligibility) verifiability. Indeed, there might be. But the definition of global verifiability does not rule out the possibility of gaps, either: any gap in the formal statement of a goal will lead to a vulnerability. That is, if the analyst forgets to include some necessary facet of verifiability when stating the formal goal, then global verifiability will not detect any attacks against that facet. Global verifiability does not guarantee a lack of gaps.

**VIII. Concluding Remarks**

When we began this work, we were studying the Juels et al. [59] definition of election verifiability. We discovered that the definition fails to detect biasing and collusion attacks. While attempting to improve the Juels et al. definition to rule out those attacks, we discovered that factoring it into individual, universal, and eligibility verifiability led to an elegant decomposition of (mostly) orthogonal properties. We later sought to apply our new definitions to existing electronic voting systems, and Helios [6] and Civitas [29] were natural choices. But they treat authentication differently—Helios outsources authentication, whereas Civitas does not—so we were led to separate our definitions into variants for external and internal authentication. We were at first surprised to discover that JCJ, hence Civitas, does not satisfy the strong definition of eligibility verifiability. But upon reflection, it became apparent that an adversary who knows the teller’s private key can easily forge ballots that appear to be from eligible voters.

Our definitions of verifiability have not addressed the issue of voter intent—that is, verification by a human that the ballot submitted by a voter corresponds to the candidate choice the voter intended to make. Adida and Neff call this property “cast as intended” [7]. Many election schemes (e.g., [42], [50], [59], [61]) do not satisfy cast as intended, because the schemes implicitly or explicitly assume that voters can themselves verify the cryptographic operations required to construct ballots. Nevertheless, schemes by Chaum [23], Neff [73], and Benaloh [11], [12] introduce cryptographic mechanisms to verify voter intent. It would be natural to explore strengthening our definitions to address voter intent.

The goal of this research is to enable verifiability of the voting systems we use in real-life, rather than merely trusting them. Research on verifiability can generalize beyond voting to other systems that must guarantee strong forms of integrity. Verifiable voting systems thus have the potential to contribute to the science of security, to democracy, and to broader society.

**Acknowledgments**

We thank David Bernhard, Jeremy Clark, Véronique Cortier, David Galindo, Markus Jakobsson, Steve Kremer, Ralf Küsters, Elizabeth Quaglia, Mark Ryan, Susan Thomson, and Poorvi Vora for insightful discussions that have influenced this paper. This work is partly supported by the European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC project CRYSP (259639), by AFOSR grants FA9550-12-1-0334 and FA9550-14-1-0334, by NSF grant 1421373, and by the National Security Agency. This work was performed in part at George Washington University and INRIA.
DEDICATION

Ben Smyth dedicates his contribution to the loving memory of Anne Konishi, 1971 – 2015. What matters most of all is the dash. We had a great time.

He writes for Christina Mai Konishi. Smile like your mother, for good fortune seeks those who smile (warau kado niwa fuku kitaru, says the Japanese proverb).

REFERENCES


[31] Véronique Cortier and David Galindo. Private communication, Nancy, France, 13th June 2013.


[44] David Galindo and Véronique Cortier. Private email communication, Summer/Autumn 2014.


