Election Verifiability with Voting Machine Adversaries

Ilana Blyakher
blyak008@umn.edu
University of Minnesota, Twin Cities
Minneapolis, MN

Calvin Roth
rothx195@umn.edu
University of Minnesota, Twin Cities
Minneapolis, MN

Samira Champlin
champ142@umn.edu
University of Minnesota, Twin Cities
Minneapolis, MN

Devon Tuma
tumax040@umn.edu
University of Minnesota, Twin Cities
Minneapolis, MN

ABSTRACT

One potential attack on an election scheme is manipulation of voting machines. There are many election schemes that provide election verifiability, but they largely rely on a secure, uncompromised voting machine. With the protocol proposed in this paper, we show how an election scheme can be extended such that it is secure even if the voting machine may not be trustworthy. We also describe how the election scheme retains coercion resistance with our protocol. In our scheme, the voter uses an interactive proof with repeated trials to ensure with a high probability that their vote is being is recorded correctly. Finally, we propose a Helios implementation that uses the interactive proof to ensure election verifiability with an insecure voting machine.

KEYWORDS

Voting, Election Verifiability, Election Security, Interactive Proofs

1 INTRODUCTION

1.1 Related Work

The original paper by Smyth, Frink and Clarkson [8], upon which we extend, outlines a framework for discussing verifiability in various aspects of an election scheme. It then examines some existing election schemes (Helios and JCJ) under these definitions of verifiability, and offers new versions for the schemes that do not meet them. Their existing definitions of verifiability remain similar to ours, as ours simply add another layer of possible verification available to voters. We outline the original definitions in Section 1.2, and add our expansion into them in Section 4.2. We also reexamine the proofs they perform on existing election schemes, Helios in particular, to identify whether the proofs still hold.

Furthermore, we utilize a paper by Josh Benaloh [3] that was referenced in the primary paper. We mainly focus on his discussion of interactive proofs, which we will examine in Section 1.3. He describes the usage and implementation of interactive proofs involving a prover and a verifier to show that some claim is true. He then shows how interactive proofs are useful for verification of elections. We can apply this to our own work to allow voters to verify with certain confidence that their vote is created fairly, even with the possibility of a tampered voting algorithm.

David Bernhard and Bogdan Warinschi write about principles and techniques used in cryptographic voting [4]. This paper provided familiarity with the cryptographic practices that are common for electronic voting to ensure that we had the background knowledge necessary to properly analyze and extend the primary paper. Bernhard and Warinschi discuss the use of cryptographic primitives in different schemes, including Helios, which we will use in a case study in Section 6.

Kremer, Ryan and Smyth [7] also look at electronic election verifiability. They provide similar definitions of verifiability as given in our primary paper, as Smyth was an author on both. However, their older definitions consist of boolean tests that can be performed on the data given from elections. We used this paper to learn of slightly different ways in which our original definitions of election verifiability could be defined.

Bräunlich and Grimm look at election verifiability as well, but their approach is entirely in the form of a bulletin board [5]. They ensure that even with the use of a bulletin board, where anyone can view all ballots, complete secrecy can still be achieved. They focus primarily on voters being able to verify that their was recorded correctly, which is the same goal as our own. However, they also do not account for the possibility of a compromised voting algorithm, in which an adversary could cause the ballot representation to look correct on the bulletin board, when the vote was actually altered when counted.

1.2 Definition of an Election Scheme

1.2.1 The Voting Algorithms. An election scheme has two classes of individuals: voters and a central system that collects the votes. The central system is commonly called a tallier. The framework introduced by Smyth, Frink and Clarkson consists of four algorithms [8]:

$$(\texttt{Setup}, \texttt{Vote}, \texttt{Tally}, \texttt{Verify})$$

Their definitions of these algorithms focus on their functionality, but not on their security. The security of the election scheme is defined in terms of different kinds of election verifiability, which we discuss in 1.2.2. Meeting these definitions requires placing certain constraints on the behavior of these four algorithms.

`Setup` takes in some security parameter k and returns public and private keys for the tallier PK_T and SK_T, and some maximum number of candidates n_C. For the purposes of the remaining algorithms in this scheme, all encryption occurs using this public key, since only the tallier ever needs to perform decryptions.

`Vote` is an encryption algorithm which returns a valid ballot or an error value if some field is not well formed. `Vote` takes as

input the number of candidates in the election n_c, the voters choice, which we represent as a number c such that $0 \leq c < n_c$, the public key of the tallier, and a security parameter. We will say that a vote is well formed if it is a valid output of some valid input to the `Vote` algorithm.

These votes are encrypted with the public key as part of the vote algorithm. The voter's vote gets put on a public bulletin board *BB*. After the voting period has ended, the tallier executes a `Tally` function which takes in the secret key, the bulletin board, the number of candidates, and the security parameters. It returns a vector of length n_c, where the ith position is how many votes candidate i got. It also returns a non-interactive proof that the tally is correct. When the tally has been announced, a voter can audit the outcome of the election by running `Verify` which takes the bulletin board, the public key, a proof that the tally is correct, the tally, number of candidate,s and the security parameter. It returns 1 is the proof is verified to be correct and 0 otherwise.

1.2.2 The Original Scheme. The original election scheme by Smyth, Frink and Clarkson [8] checks individual, universal, and eligibility verifiability for both external and internal authentication using the previously defined experiments that output 0 if the adversary loses the experiment, and 1 if they win the experiment.

An election scheme has individual verifiability if a voter can uniquely identify their ballot on a display of all ballots and verify that it has been recorded. In a scheme with external verification, the algorithm asks the adversary to compute two candidate choices β and β', such that the ballots computed by `Vote` are equal. Individual verifiability is achieved if the adversary cannot find a collision where the two votes result in the same ballot. However, this algorithm is only effective if the correct `Vote` algorithm is run. If the experiment is for a scheme with internal authentication, then the adversary also has access to an oracle, which is used to model the adversary corrupting voters and learning their private credentials.

In order for an election to be universally verifiable, anyone must be able to check that a tally is correct based on the recorded ballots. In an election scheme with external authentication, the experiment challenges the adversary to make `Verify` incorrectly accept a vote. In the algorithm, the adversary must create a bulletin board where the correct tally is Y, and give the bulletin board a tally of X. If X is not equal to Y and `Verify` accepts the tally X, then the adversary wins. If the adversary cannot win, then the election has universal verifiability. If the election scheme has internal authentication, the voters are registered and their credentials are used in the rest of the experiment. The experiment explicitly handles re-voting by tallying only ballots that are constructed with a private credential that was not used to construct any other ballot on the bulletin board.

For the election scheme to have eligibility verifiability, anyone must be able to check that every tallied vote was cast by an authorized voter. In an election scheme with external authentication, authentication is done by a third party. While the third party may convince itself that all tallied ballots are authenticated, it cannot convince the other parties, so eligibility verifiability is not achievable. In a scheme with internal authentication, the experiment challenges the adversary to produce a ballot under a private credential that the adversary does not know. If the adversary can produce an authentic ballot that is the output of `Vote` on an authorized credential, the credential belongs to a voter that has not been corrupted by the adversary, and the ballot is not revealed, then the adversary has won. The election scheme with internal authentication has eligibility verifiability if the adversary cannot win.

An election scheme with external authentication has election verifiability if the adversary cannot win the experiments for individual and universal verifiability. If the election scheme has internal authentication, then it has election verifiability if the adversary cannot win the experiments for individual, universal, and eligibility verifiability

1.2.3 The Verifiablilty Definitions. Smyth, Frink and Clarkson introduced three flavors of verifiability that a secure voting system should have. We will now provide some formal definitions for the more relevant ones with which we are working. The first is universal verifiability. As discussed, this is the property take anyone can check if the tally is actually correct. The `Verify` algorithm, which returns whether the tally is correct, is meant to ensure the property of universal verifiability. We can also define what it means for a system to be universal verifiable with the experiment $Exp - UV - EXT(\prod, A, k)$ by Clarkson and Compev [8]

$Exp - UV - Ext(PK_\tau, SK_\tau, A, k)$:
  $Y \leftarrow correct - tally(PK_\tau, BB, n_c, k)$
  $X \leftarrow A(PK_\tau, BB, n_c, k)$
  if $Y \neq X \wedge Verify(PK_\tau, BB, n_c, X, P, k) = 1$ **then**
    return 1
  else
    return 0

Where $correct - tally$ is a theoretical algorithm that always returns the correct tally on a given input (this algorithm is purely theoretical, but is useful in this analysis). To satisfy universal verifiability, no adversary should be able to win this game with a non-negligible probability.

The second property we have is the property of eligibility verifiability, which is the ability for anyone to check that every vote came from an authorized voter. This means that every vote that was counted (there can be improper votes that are not counted in the tally) can be authenticated. This property is only achievable if an election scheme performs voter authentication internally [8]. Because we focus on Clarkson's external authentication model, this property will be of little importance in our work.

The third property, which is the main subject of our work, is individual verifiability. Individual verifiability is the ability of any voter to look at all of the encrypted votes on the bulletin board and be able to verify with confidence that their vote was recorded, and that it was recorded correctly. The authors define a game for this property as the Exp-IV-Ext game which is analogous to a collision finding game.

$$
\begin{aligned}
&Exp - IV - Ext(PK_\tau, SK_\tau, A, k):\\
&\quad (PK_\tau, n_c, B, B') \leftarrow A(k)\\
&\quad b \leftarrow VOTE(PK_\tau, n_c, B, k)\\
&\quad b' \leftarrow VOTE(PK_\tau, n_c, B', k)\\
&\quad \textbf{if } b = b \wedge b \neq \perp \wedge b' \neq \perp \textbf{ then}\\
&\qquad \text{return } 1\\
&\quad \textbf{else}\\
&\qquad \text{return } 0
\end{aligned}
$$

To satisfy individual verifiability, no adversary should be able to win this game with a non-negligible probability. Clarkson and Compev show that Universal verifiability and individual verifiability are orthogonal properties of a voting system which means that they are independent properties of a system and that neither implies the other [8].

Coercion Resistance. Another important property of election schemes discussed by Smyth, Frink and Clarkson is coercion resistance. A more thorough discussion of this is given by Juels, Catalano, and Jakobsson [1], and this is were we mainly draw from. Coercion resistance is the property that no voter can prove to a third party which candidate they voted for. A rigorous definition of this property for our version of an election scheme is difficult, but trying to maintain this property will influence the approach we take to extending the election scheme. While Helios is not completely coercion resistant, our extensions will work to not weaken the degree of coercion resistance already present.

1.3 Interactive Proofs

Interactive proofs play a notable role in our contribution. In an interactive proof system, one party, the verifier, can ask the other party, the prover, to answer a question that it could only answer if the prover really had some piece of information. In a zero-knowledge system this must be done without revealing the specific piece of information. It uses repeated trials, with the purpose being to make it increasingly unlikely that the prover is faking their answers.

Two important properties for interactive proofs are soundness and completeness [2]. Completeness says that if a statement is true, then an honest verifier will always be convinced that the statement is true. Soundness says that if it is not true, then an honest verifier will be convinced it is true with only small probability, even if the prover does not follow the protocol.

2 MOTIVATION

Voting could be made more accessible with the added convenience of online possibilities. However, while online voting may be more convenient, it also leaves more room for security failures. Verification is one way to mitigate these security failures. This includes voters' verification of their own vote, anyone's verification of proper vote computation, and anyone's verification of voters' authentication. Correctness of such election verifiability is necessary. Without a method to verify if an election has not been tampered with, there is no way of knowing if the results of an election are accurate. Smith, Frink, and Clarkson discuss a definition of verifiability that should ensure such accuracy by way of verification. However, they point out one potentially large shortcoming of their definition that could invalidate their findings if not taken into account. We will examine this shortcoming and alter the current definition by Smith, Frink, and Clarkson to fix it while still preserving old security properties.

2.1 The Flaw

The definition of election verifiability discussed by Smith, Frink, and Clarkson fails to account for cases in which a voter does not use the correct `Vote` algorithm. This could happen if an adversary somehow replaces the `Vote` algorithm with a `Vote'` algorithm, perhaps by altering some hardware or software related to `Vote`. If this were to happen, verification would be unreliable for a voting scheme that was deemed verifiable under Smith, Frink, and Clarkson's definitions. As a simple example, a voting machine could give the same ballot to multiple voters by choosing non-random nonces in their encryption. Then the tallier could collude by placing only one of these identical ballot on the bulletin board. From the perspective of each voter, they think they can see their vote and therefore think it has been counted. However their votes have been artificially suppressed into a single vote.

Smith, Frink, and Clarkson point out this flaw in their definition of individual verifiability. They suggest potential avenues for fixing this, but opt not to explore the issues. One big factor in this vulnerability is that voters can run `Vote` on personal machines, which are usually very vulnerable. The most natural way to fix this is to have the `Vote` algorithm be run by select voting machines and/or servers that can be trusted.

The problem remains that voters may still not trust these machines to run correctly, since they could still be vulnerable to adversaries, and may also now vulnerable to internal manipulation by the body setting up these machines. The key issue is that while `Tally` has the `Verify` algorithm to audit it, `Vote` has no such algorithm, and the nature of the election scheme makes it hard to directly introduce one.

The way we handle this is to introduce an interactive proof to the voting scheme, in which voters may become confident in the validity of their ballots. This approach is partially inspired by the work of Benaloh [3] on interactive proofs in electronic elections.

We pursue a scheme in which auditing in this way requires some computational power on the voters part, but voting in general does not (i.e. participating in this proof is optional). Anyone can participate if they have the ability to perform computation, but it is not required of any voter.

In a real world scenario, we may expect the number of people who choose to audit in this way would be small. we will therefore set up our scheme such that an adversary cannot distinguish auditors from non-auditors until the adversary has already committed to some choices. In this context, we will analyze the chance of an unnoticed adversary attack if only n of m voters participate in this auditing, and show that it is small even for small n. Thus, even voters who do not participate in an interactive proof have some degree of security.

3 APPROACH

3.1 Technical Issue

The specific issue with the system given by Smith, Frink and Clarkson is the individual verifiability component of election verifiability [8]. Individual verifiability is defined by an experiment `Exp-IV-Ext` which challenges an adversary to produce two candidates that lead to identical ballots when used as inputs to the `Vote` procedure. An election scheme is then called individually verifiable if no adversary can win with non-negligible probability. This definition precludes the possibility that a voter will be unable to uniquely identify their ballot on the bulletin board.

This still leaves the possibility of a clash attack on the election scheme. The definition only ensures that a voter can uniquely identify their ballot if they ran the election scheme's prescribed voting algorithm `Vote`. If their computer was altered by some adversary to run some other `Vote'`, then the definition of individual verifiability does not guarantee that their ballot will be uniquely identifiable on the bulletin board.

Naive Fix

The naive approach to this problem is that votes should be accompanied by proofs that they are accurate votes for specific candidates, and that these votes are properly randomized. However, this weakens coercion resistance, especially if such a proof is hard to forge (which it must be if it is going to prevent misbehavior by the voting machines). This approach is therefore not sufficient for solving the problem.

Instead, we opt to adjust the definition of an election scheme to allow interactive proofs between voters and the `Vote` algorithm, so the voters can verify the validity of ballots they receive. Then, we are able to introduce a stronger definition of individual verifiability that prevents these kinds of clash attacks. We also set up the protocol such that the final vote a voter submits is one for which they have no proof, which serves to help preserve coercion resistance.

The specific idea is that a voter will ask for some large number of votes, then randomly choose one as their actual vote. Then they request proofs on all the other ballots, and if they are valid (along with a few other conditions), the voter will have their chosen vote signed and submitted. The protocol will be defined more carefully in Section 4.3.2.

We will consider any case ih which a voter discovers an adversary manipulating the integrity of the election as a success for the voter. We choose not to focus on what a voter should do after discovering such a manipulation, as it is just as much a social and political issue as a cryptographic one. This approach is consistent with both Smyth, Frink and Clarkson [8] and Benaloh [3].

4 FIX

4.1 Definition of Vote Algorithm as a Protocol

Because the new protocol we wish to define requires new algorithms to be run, we start by altering the definition of an election scheme with external authentication to be a 6-tuple
(`Setup, Vote, Tally, Verify, Sign-Ballot, Prove-Ballot`)
where the first four elements are modified from the previous version, and the last two elements are new probabilistic polynomial time algorithms:

Prove-Ballot *denoted* $(\beta, I) \leftarrow \textbf{Prove} - \textbf{Ballot}(k, PK_t, b)$ *is executed by a voting machine. It takes as input the security parameter* k*, public key of the tallier* PK_T*, and a ballot* b*. It returns all the information* I *that the voter would need to have to check that ballot* b *corresponds to candidate* β*. It will refuse to provide a proof on a ballot that has been run through the* ***Sign-Ballot*** *algorithm.* ***Prove-Ballot*** *opens up the possibility of coercion, as it gives voters proof that they voted for a certain candidate.*

Sign-Ballot *denoted* $(b') \leftarrow \textbf{Sign} - \textbf{Ballot}(k, PK_T, b)$ *is executed by a voting machine. It takes as input the security parameter* k*, public key of the tallier* PK_T*, and a ballot* b*. It returns a signed ballot constructed from* b*, that still identifiably contains* b*. It refuses to sign if the ballot has previously been decrypted by* ***Prove-Ballot****.* ***Sign-Ballot*** *prevents issues related to coercion by not allowing votes for which a proof exists to be counted.*

Note that both of these algorithms require the machine running them to maintain some kind of internal state. These two algorithms are the only algorithms in the election scheme that require an internal state. They, along with `Vote`, are the only algorithms that are run by an external voting machine, rather than a central server. For the purpose of our analysis, we treat these voting machines as adversaries to a voter (so a voter has no guarantees about any algorithms they run), but we treat the central server as a trusted party that behaves correctly. It may seem strange to entrust the machine running `Sign-Ballot` and `Prove-Ballot` to make them mutually exclusive when we are treating the voting machines as adversaries, however we point out that if an adversary did not do this, they would also be revealing to the voter that the election lacks integrity by revealing that the voting machine isn't acting properly. We can therefore disregard this possibility.

We should also note that the information I returned by `Prove-Ballot` is not an immediate decryption, but rather a proof of encryption. For example, I could be a nonce that was used as an encryption key, so that the voter could later run an the encryption with their candidate, the nonce, and the public key and verify they receive back their ballot. The reason for this is that the scheme is set up so that while voting machines can encrypt ballots, they are unable to perform decryptions (and in fact no one but the centralized server can efficiently decrypt any ballots).

The `Setup` and `Vote` algorithms remain unchanged at the base level, aside from some potential changes to ensure coercion resistance. `Tally` and `Verify` must now expect signed ballots produced by `Sign-Ballot`, not regular ballots produced by `Vote`. The specific requirements for these algorithms is that they must be changed enough to meet the updated security definitions we provide below.

`Vote` was previously expected to perform some kind randomization on its own in order to ensure injectivity of the `Vote` algorithm, since we want to prevent generation of identical ballots. Since we now plan to not trust the voting machines behavior and wish to audit them, we expect voters to also provide some random values used by `Vote`. The `Vote` algorithm therefore now takes one extra input s, and is required to include s in such a way that changing s in any way completely changes the ballot (e.g. including s inside of a secure symmetric encryption). The voting protocol we introduce

later will ensure that the Vote algorithm respects this random value. Thus voters can ensure that randomization is still involved in their ballot.

4.2 Old Definitions of Security Updated for New Election Scheme Model

The old definitions of security must be updated to account for these new algorithms. While the changes to the definitions are mostly trivial and mainly involve including the Sign-Ballot algorithm, we include them here for completeness. We also mention ways in which our new algorithms may affect security, and comment on what constraints these properties place on Sign-Ballot and Prove-Ballot.

4.2.1 Correctness. We first extend the definition of correctness provided by Smyth, Frink and Clarkson [8]. There exists a negligible function μ such that for all security parameters k, integers n_B and n_C, and choices $\beta_1 \dots \beta_{n_B} \in \{1, \dots, n_C\}$, it holds that if Y is a vector of length n_C whose components are all 0, then

$$
\begin{aligned}
&\Pr[(PK_\tau, SK_\tau, m_B, m_C) \leftarrow \text{Setup}(k) : \\
&\quad \textbf{for } 1 \le i \le n_B \textbf{ do} \\
&\qquad b_i \leftarrow \text{VOTE}(PK_\tau, n_C, \beta_i, k) \\
&\qquad b'_i \leftarrow \text{Sign-Ballot}(b_i) \\
&\qquad Y[\beta_i] \leftarrow Y[\beta_i] + 1 \\
&\quad BB \leftarrow \{b'_1 \dots b'_{n_B}\} \\
&\quad (X, P) \leftarrow \text{Tally}(SK_\tau, BB, n_C, k) \\
&\quad n_B \le m_B \wedge n_C \le m_C \Rightarrow X = Y] > 1 - \mu(k)
\end{aligned}
$$

An election scheme that previously satisfies this property could fail to satisfy it if Sign-Ballot or Tally do not properly agree. The simplest way to preserve this is to have Sign-Ballot still contain the old ballots internally somewhere, and then use those to reuse the old Tally implementation.

4.2.2 Individual Verifiability. We extend the definition of individual verifiability provided by Smyth, Frink and Clarkson [8]. Individual verifiability will eventually be subsumed as part of a stronger definition given in section 4.3.2, but we provide an updated version of the definition here for completeness. As in the paper by Smyth, Frink and Clarkson, we define an experiment Exp-IV-Ext as

$$
\begin{aligned}
&\text{Exp-IV-Ext}(\Pi, A, k) = \\
&\quad (PK_\tau, n_C, \beta, \beta') \leftarrow A(k) \\
&\quad b \leftarrow \text{Vote}(PK_\tau, n_C, \beta, k) \\
&\quad b' \leftarrow \text{Sign-Ballot}(b) \\
&\quad b_1 \leftarrow \text{Vote}(PK_\tau, n_C, \beta', k) \\
&\quad b'_1 \leftarrow \text{Sign-Ballot}(b_1) \\
&\quad \textbf{if } b' = b'_1 \wedge b' \neq \perp \wedge b'_1 \neq \perp \textbf{ then} \\
&\qquad \text{return } 1 \\
&\quad \textbf{else} \\
&\qquad \text{return } 0
\end{aligned}
$$

We say that an election scheme is individually verifiable if no adversary can win this game with greater than negligible probability. The biggest issue that could be introduced with this is if Sign-Ballot is not collision resistant like the old Vote. The simplest fix for this is that the output of Sign-Ballot should contain the whole of its input as a sub-string. Then, since Vote was collision resistant, this will be too.

4.2.3 Universal Verifiability. The definition of universal verifiability does not change from the original definition, aside from the fact that it must now use the updated Verify algorithm.

4.2.4 Eligibility Verifiability. Eligibility verifiability is the concept that anyone must be able to check that every tallied vote was authorized. As with the paper by Smyth, Frink and Clarkson, this is only possible if the election scheme operates with an internal authentication model [8]. To keep this work focused, we opt to concentrate our analysis on external authentication models only, and leave internal authentication and eligibility verifiability as future work.

4.3 Definitions of Security for New Election Scheme

The purpose of the introduction of the two algorithms Sign-Ballot and Prove-Ballot is to allow the voter and voting machine to participate in an interactive protocol, in which the voting machine proves to the voter that their ballot corresponds to the candidate for whom they voted. However, this introduces more potential security issues into the election scheme. Therefore, we define several new election security properties that, if satisfied, can be used to show that an election scheme is still secure under this new model.

4.3.1 Coercion Resistance Under Multiple Voting Machine Attacks. Because the Prove-Ballot algorithm is not required to be zero-knowledge, the introduction of the Prove-Ballot algorithm potentially conflicts with coercion resistance, since a voter could theoretically submit a vote for which they have a proof that corresponds to a particular candidate.

The introduction of the Sign-Ballot algorithm as an extra step is designed to prevent this and preserve coercion resistance. Because we disallow voting machines from signing ballots for which they previously provided proofs, and because Tally requires votes to be signed, a voter can no longer submit a ballot for which they have a

proof. However, this relies on the fact that a voter can only ask the voting machine that produced a given ballot to prove or sign that ballot.

We therefore define the multiple voting machine coercion resistance experiment $MVM-CR-Ext(\Pi, A, k)$, where Π is an election scheme, A is an adversary, and k is a security parameter by:

$MVM-CR-Ext(\Pi, A, k)$:
$(PK_t, n_c, \beta) \leftarrow A(k)$
$b \leftarrow VOTE(PK_t, n_c, \beta)$
$I \leftarrow A_{Prove-Ballot}(b)$
$b' \leftarrow A_{Sign-Ballot}(b)$
if I a valid proof for b **then**
return 1
else if b' a valid signed ballot **then**
return 1
else
return 0

This experiment challenges the adversary to either give an encryption proof for or sign a ballot produced by a different honest voting algorithm. If an adversary can only win a negligible amount of time we say the election scheme Π is coercion resistant under multiple voting machine attacks. If this property holds and the election scheme was originally coercion resistance before modification, then the modified election scheme will be coercion resistant in the general sense.

It is worth noting that the ballot is generated by a correct voting algorithm, when we are treating the machines running `Vote` as adversaries. Theoretically two voting machines could collude to break this property. However, that collusion would need either one voting machine to produce non-random ballots that the other would expect, or to communicate with the other voting machine. In the first case scenario, this could be detected by our stronger definition of individual verifiability, and the second can be mitigated by preventing network communication between voting machines. We can therefore ignore this possibility in our analysis, and assume that only one of the two voting machines is an adversary.

4.3.2 Individual Verifiability with Adversarial Voting Machines. As stated before, these two new functions allow a protocol between the voter and the voting system, in which the voter treats the voting algorithm as an adversary that may attempt to secretly modify their vote. The voter has the option to to repeatedly request any number of ballots and then ask for proofs on any number of those ballots, by calling `Vote` and then `Prove-Ballot`. When the voter is confident enough in the correctness of the machine, it calls `Sign-Ballot` on one of the ballots.

Note that the voter makes votes sequentially, so the voting machine is not aware of the number of votes a voter will request in advance of producing them. This prevents the voting machine from deciding whether to try and cheat based on how closely its output will be examined.

The voter then submits the signed ballot and their authentication to the voting server. From this point on, the election scheme continues parallel to how it did in the previous scheme, the only difference being that the server now checks if ballots are properly signed by the `Sign-Ballot` algorithm.

We define the protocol more rigorously as

1 The voter requests n ballots using `Vote`. The voter gives random nonces to each call to `Vote` as their random input.
2 The voting machine produces the ballots, ensuring none of the votes are duplicates.
3 The voter selects some ballot b_k with $1 \leq k \leq n$ as their final ballot.
4 The voter asks for proofs on all ballots b_j with $j \neq k$.
5 The voting machine returns valid proofs for all such ballots, assuming it is able.
6 The voter asks for a signature on ballot b_k.

The statement the voter is verifying will be "The ballot that I had signed corresponds to the candidate I voted for, and is correctly randomized based on my seeding". The voter is convinced this is true if the voting machine answered all requests, all the proofs for the other ballots are true in the context of the seeds they provided, and none of the ballots were identical. Otherwise, the voter will be convinced it is false.

Theoretically, it is possible for a voter to interleave calls to `Vote` and `Prove-Ballot`, but this does not provide them any advantage in determining whether the statement is true. Therefore, in order to simplify our analysis, we opt to assume that the voter performs all `Vote` calls first. A voter that is not interested in participating in the auditing process for whatever reason simply skips the step of calling `Prove-Ballot`, instead just signing a vote of their choice and submitting it.

We then say that the election scheme is Individually Verifiable with Adversarial Voting Machines (IV-AVM) if it is individually verifiable in the regular sense of Section 4.1.2, and if the protocol defined above is both sound, complete, and zero-Knowledge in the sense that the voter learns nothing about the vote they had signed. However, we give no zero-Knowledge constraints to the voter's knowledge about the votes they did not have signed.

Soundness here means that if the signed ballot the voter received is manipulated, they will only be convinced it is honest with some very small probability, even if the voting machine did not follow the protocol.

Completeness here means that if the voting machine followed the protocol correctly, the voter will be convinced of the integrity of their ballot all of the time.

4.3.3 Correctness of Prove-Ballot algorithm. The definition of IV-AVM is somewhat difficult to work with in practice, because it requires showing three properties on top of just showing individual verifiability. As such, we provide a definition of correctness that will imply this stronger definition.

First, we define an experiment that can be used as a heuristic to show that the protocol is sound. We define the existential unforgeability of the ballot encryption experiment $EUF-BP-Exp(\Pi, A, k)$ by:

$EUF - BP - Exp(\Pi, A, k)$:
$(n_c, \beta, \beta') \leftarrow A(k)$
$b \leftarrow A_{Vote}(PK_t, n_c, \beta, k, s)$
if b is not well formed ballot **then** :
return 0
$I \leftarrow A_{Prove-Ballot}(b)$
if I proves b is a vote for β' with seed s **then** :
return 1
else :
return 0

This experiment challenges the adversary to generate a well-formed ballot for one candidate, and a proof that this ballot is a ballot for another candidate. If an adversary cannot do this with more than negligible probability, we say that the election scheme has existential unforgeability of ballot encryptions. We will later show that this property is sufficient to prove that an election scheme's voting protocol is sound.

5 GENERIC RESULTS

While most proofs are only possible in the context of a specific election scheme implementation such as Helios, there are some generic results that we can include now before defining our new version of Helios.

5.1 EUF-BP Implies Soundness

Assume that an election scheme does in fact have the EUF-BP property, but that the voting protocol was not a sound interactive proof protocol. Therefore, by the definition of soundness, some adversary must be able to, with large probability, get a voter to accept that the signed ballot they received was honest when it in fact was not.

Assume that during the protocol, the voter requests m ballots from `Vote` in total. If any ballots were identical, then a voter following the protocol would necessarily reject, so since the adversary is successfully tricking them we can assume that all the votes must be distinct. Also, if any ballots were not well formed, the voter would reject, so we can assume that is not the case either.

An honest voter following the protocol would always call prove-ballot on all but one of the m ballots they receive. Then assume that the voting machine was to create multiple manipulated ballots. By the pigeonhole principle, the voter will ask for a decryption of at least one manipulated ballot. Since we assumed the elections scheme was EUF-BP, the probability of the voting machine successfully producing a proof for such a ballot must be negligible. However, we assumed that the voting machine had non-negligible chance of success at manipulating ballots, so this cannot happen either. Thus, the voting machine must be manipulating exactly one ballot.

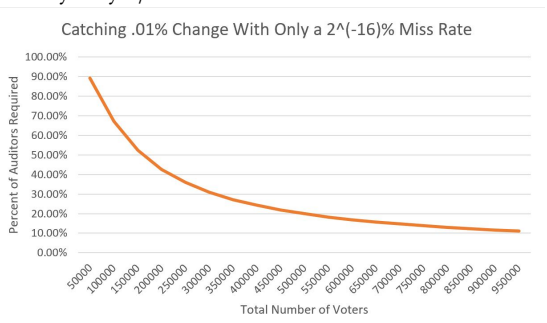
Since the voter chooses which ballot to have signed uniformly and randomly, there is a $1/m$ chance that the voter will choose to have the manipulated ballot signed. Then, the chance that the voter has their vote manipulated is at most $1/m$. However, $1/m$ gets small as m gets larger, and we assumed that the voter was having their vote manipulated with large probability. Therefore this is a contradiction, and the must not have had the EUF-BP property. By contrapositive, we have shown that the EUF-BP property implies soundness of the election scheme's voting protocol as an interactive proof system.

5.2 Required Number of Auditors

We now want to consider how many of the voters in an election scheme must perform auditing in order to be highly likely to detect manipulation by an adversary. Assume that we have an election scheme that satisfies individual verifiability with adversarial voting machines. We first consider a very ideal model of the situation. Assume that v total voters will cast ballots, and an adversarial voting machine decides to modify some number k of them. Because of the IV-AVM property, an adversary will be caught with odds about $(n-1)/n$ if someone that chooses to audit the ballots receives one of the adversaries modified ballots, where n is the number of votes they request in the auditing process. For simplicity, assume n is large enough that this probability is essentially 1. Assume that a total of w voters will participate in this auditing. The chance of the adversary succeeding is the chance that they distribute the ballots among only the non-auditors, which is exactly

$$\frac{(v-w)Ck}{vCk} = \frac{\frac{(v-w)!}{(v-w-k)!k!}}{\frac{v!}{(v-k)!k!}} = \frac{(v-w)!(v-k)!}{(v-w-k)!v!}$$

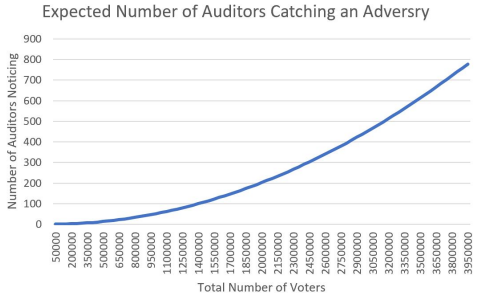
Using this equation as a model for a python script, we can create the following figure which shows how many auditors would be required to notice a change to 1/10000 of the votes with failure probability only $1/2^{16}$.



One striking thing about this is that the number of auditors required is decreasing logarithmically as the number of voters increases. This may be surprising, since it may appear that if the percent of votes being altered stays constant, then the percent of auditing voters should stay constant. The fact that this is not the case suggests that this system has large scalability, as the size of the election increases, the vulnerability actually goes down, not up. This behavior is reflect more generally in the equation as well: if v, w, k are all scaled by some positive constant, the value actually goes down (i.e. chance of adversary success decreases). This is simply due to the fact that $\frac{(cn_0)!}{(cn_1)!} > \frac{n_0!}{n_1!}$ if $n_0 - n_1 > 0$ (to see this, note that the larger factorial grows by a larger quantity of larger factors).

Another thing to consider, however, is how many voters will catch such an adversary. In practice, a single voter catching an adversary may not be enough to convince other voters; a larger threshold may be needed. We are therefore interested in calculating the expected number of voters that will detect this same attack. Define an indicator function for each voter by χ_v is 1 if voter v catches an adversary altering their own vote and χ_v is 0 otherwise. If a particular voter v does not do any auditing, then $E[\chi_v] = 0$ since χ_v is always 0. On the other hand, if this particular voter does do auditing, we can calculate $E[\chi_v] = 1 * Pr(\chi_v = 1) = Pr(A \text{ targets } v)$.

Assuming that the adversary is targeting voters randomly, this gives $E[\chi_v] = 1 - \prod_{i=0}^{k}(1 - \frac{v-1-i}{v})$. Then by the linearity of expectation, the total expected number of voters that will detect an attack is $wE[\chi_v] = w(1 - \prod_{i=0}^{k}(1 - \frac{v-1-i}{v}))$. Using this as a model, the following figure shows the expected number of auditors that would notice an adversary assuming 1% of voters are auditors and an adversary affects 0.01% of votes.



The effect is somewhat less pronounced, but its still clear that larger elections make things harder for an adversary. As the total number of voters rises, the expected number of auditors catching the adversary is increasing as well. This again suggests strong scalability of the system.

6 CASE STUDY: HELIOS

We can apply these definitions to extend Helios [1] to a new version, Helios*. Of the 6-tuple of algorithms in our voting schemes, the original four, `Setup, Vote, Tally, and Verify`, will remain mostly unaltered from their definitions by Smyth, Frink and Clarkson [8]. `Setup, Verify` will be completely unchanged, while `Vote, Tally` will see small alterations.

The original implementation of `Vote` in Helios and Helios'16 uses an Elgamal encryption scheme because it requires the ballot encryption to be homomorphic, and Elgamal is known to be homomorphic [6]. The encryption essentially corresponds to $E_{PK_T}(v)$ where v is the vote. Now we update this to be $F_{PK_T}(E_{PK_T}(v)||N||s)$ where F is some other potentially non-Elgamal encryption scheme, s is the voter provided random seed, and $N \in_R \{1, \cdots, 2^k\}$ is some random k bit nonce, generated with the voter provided seeding. The voting machine must track this nonce in an internal state as corresponding to this ballot for future use in `Sign-Ballot`.

Note that F and E are both encrypted with PK_T, but are different encryption schemes. For simplicity, we just assume that the output PK_T of setup is now a concatenation of the key for each encryption scheme, with the convention that F_{PK_T} and E_{PK_T} use the appropriate half of the concatenation for their encryption.

We define the `Sign-Ballot` algorithm of Helios* to take an unsigned encrypted ballot $B = F_{PK_T}(E_{PK_T}(v)||N)$ which has not been decrypted, the public key of the tallier, and the security parameter k. It returns $B||N = F_{PK_T}(E_{PK_T}(v)||N)||N$, where N is the nonce that was used to construct the ballot. Note that only the voting machine that created the ballot B will have knowledge of N to use in the signing.

`Prove-Ballot` in Helios* takes the security parameter k, the public key of the tallier PK_t, and a ballot $B = F_{PK_T}(E_{PK_T}(v)||N)$. It will return $v||N||R$, a combination of the nonce N used to construct the ballot, along with v the vote that was encrypted, and R which is any random values chosen by the symmetric encryption schemes E and F. For example, R must contain the random y value that is chosen when making an Elgamal encryption. Returning v is necessary because v necessarily has some randomness beyond just candidate choice in order to preserve injectivity of votes (otherwise two voters could receive identical ballots, which would negate individual verifiability). Therefore even though the voter knows the candidate for whom they voted, they do not know v. Note that knowledge of v and N is enough to verify that the ballot truly corresponds to the candidate a voter thought it did, by running the encryption personally with PK_T and these values, and checking that the output is in fact B.

At a high level, `Tally` in the original Helios and Helios'16 takes a set of encrypted ballots $E_{PK_T}(v_1), ..., E_{PK_T}(v_k)$, combines them homomorphically to get

$$E_{PK_T}(v_1) \cdot \ldots \cdot E_{PK_T}(v_k) = E_{PK_T}(v_1 \cdot \ldots \cdot v_k)$$

and then finally decrypts this final piece to get the combination of all the votes. Doing this prevents leaking any information about how certain ballots or subsets of ballots were voted, only the final tally is revealed. Our modified approach takes a list of the encrypted and signed ballots produced by the new vote algorithm $F_{PK_T}(E_{PK_T}(v_1)||N_1)||N_1', ..., F_{PK_T}(E_{PK_T}(v_k)||N_k)||N_k'$. It first decrypts the outer F encryption, and checks that $N_j = N_j'$ for all $1 \leq j \leq k$. It now has access to the internal $E_{PK_T}(v_1), ..., E_{PK_T}(v_k)$, and performs the tally as before.

6.1 Coercion Resistance in Helios*

The original election scheme has coercion resistance in Helios. However, with our addition of `Prove-Ballot`, we open up the possibility of a vulnerability of coercion resistance. `Sign-Ballot` is implemented to ensure that a ballot that has been proved cannot also be signed. As long as `Sign-Ballot` cannot be forged, then the adversary cannot win the $MVM - CR - Ext(\Pi, A, k)$ game, which proves that there is no vulnerability from introducing `Prove-Ballot` with `Sign-Ballot`.

Next, we prove that it is hard to forge `Sign-Ballot`. We can prove this by contradiction. Assume that we can find some N' and v' where $N \neq N'$ and $v \neq v'$ such that $F(E(v)||N) = F(E(v')||N')$. But note that the key to encrypt v to $E(v)$ is public, so we can then compute $E(v)||N, E(v')||N'$. Since $N \neq N'$, we must have $E(v)||N \neq E(v')||N'$. This means that we have found a collision for the encryption scheme of F. Since F is a secure symmetric encryption, it is injective, so there cannot be a collision. Thus we

[1] https://github.com/benadida/helios

have a contradiction. Therefore, `Sign-Ballot` is unforgeable. Since it is unforgeable, no adversary can win the $MVM{-}CR{-}Ext(\Pi, A, k)$ game. Therefore, the election scheme is coercion resistant in Helios*.

6.2 Helios* is IV-AVM

As we defined before, Individual Verifiability with Adversarial Voting Machines requires showing three things: regular vndividual Verifiability, as well as soundness and completeness of the voting protocol as an interactive proof system. We now show that Helios* as previously defined satisfies these three properties.

6.2.1 Helios is Individually Verifiable.* We would like our new version of Helios* to make ballots such that an adversary cannot convince two individuals that one ballot belongs to both of them with a non-negligible chance of success. We assume that F is a secure encryption scheme such that there is a collision if and only if $E(v)||N == E(v')||N'$. This can only happen if $N = N'$ which happens only with chance $\frac{1}{2}^k$ where k is the length of N and if $E(v) = E(v'), v = v'$ which happens with probability $\frac{1}{2}^{k'}$ where k' is v and v'. So there is a $\frac{1}{2}^{k+k'}$ chance that this happens.

We can now use this property to show that Helios* is individually verifiable. Suppose an adversary A could beat the $Exp_I V{-}Ext$ game with a non-negligible probability. This means we can make ballots b and b' such that $b' = b$ but where b and b' were made by two different set of arguments to `Vote`. If there were such an adversary A, then A could be used to find collisions in our `Vote` algorithm. However, we have shown that this cannot be done, so A cannot win the $Exp{-}IV{-}Ext$ game, meaning our scheme maintains individual verifiability.

6.2.2 The voting protocol in Helios is sound.* We start by showing that an adversary can only win the $EUF - BP$ game (defined in Section 4.3.3) if they can forge `Prove-Ballot`. The only way the adversary can win the game is if they can find a proof that b is a vote for β'. To do this, the adversary must find some N', v' such that $F(E(v)||N||s) = F(E(v')||N'||s)$, where N, v is the proof of b. The adversary must be able to forge `Prove-Ballot` in order to win the game.

Next, we prove that it is hard to forge `Prove-Ballot`. We can prove this by contradiction. Assume that we can find some N' and v' where $N \neq N'$ and $v \neq v'$ such that $F(E(v)||N||s) = F(E(v')||N'||s)$. Note that the key to encrypt v to $E(v)$ is public, so we can then compute $E(v)||N||s, E(v')||N'||s$. Since $N \neq N'$, we must have $E(v)||N||s \neq E(v')||N'||s$. This means that we have found a collision for the encryption scheme of F. Since F is a secure symmetric encryption, it is injective, so there cannot be a collision. Thus we have a contradiction. Therefore, `Prove-Ballot` is unforgeable. Since it is unforgeable, no adversary can win the EUR-BP game.

Since we have previously shown that EUF-BP implies that the voting algorithm is complete, and we have just shown that Helios* satisfies EUF-BP, Helios* therefore has a voting protocol that is complete.

6.2.3 The voting protocol in Helios is complete.* We simply need to show that a voting machine acting honestly and following the protocol will always convince an honest voter that the honest ballot they had signed is not manipulated. First, an honest voting machine will always answer all requests. Secondly, an honest voting machine that ran `Vote` will have access to v, N and store them in its internal state, and then can always produce them back given a call to `Prove-Ballot`. Finally, a voting machine following the protocol will not produce duplicate ballots based on step 2 of the protocol. Therefore all the requirements for the voter to accept the statement as true are met, and the voter will accept it as required. Thus the protocol is complete as desired.

7 FUTURE WORK

We suggest multiple ways to extend this paper for future work. The extension done here still needs to be implemented, and can be applied to other voting systems like JCJ and Cervitas. Another limitation of our paper is that we do not have the voter take action after they discover that their vote algorithm has been altered. Although we propose a way to find if the voting machine is insecure, we do not show how to recover the scheme from this. Additionally, we only consider a system with external authentication. For a system of internal authentication, eligibility verifiability also must be considered. Another limitation of our work is that we do not consider how this may scale once implemented. There is future work to be done to consider how this can be used in an actual election.

8 CONCLUSION

We have shown how the definitions of election verifiability given by Smyth, Frink and Clarksonby [8] can be extended to account for cases in which the `Vote` algorithm is compromised. By using interactive proofs, the voter can be certain that their vote has been recorded correctly even if there is an adversarial voting machine. In addition, the election scheme we propose enforces coercion resistance. We have also described an updated implementation of the Helios election scheme that makes use of our interactive proof. With our verifiable voting scheme, which we can use to protect against adversarial voting machines, electronic voting can be safely implemented.

REFERENCES

[1] Dario Catalano Ari Juels and Markus Jakobsson. 2002. Coercion-Resistant Electronic Elections.
[2] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach.* Cambridge University Press.
[3] Josh Benaloh. 2006. Simple Verifiable Elections.
[4] David Bernhard and Warinschi Bogdan. 2013. Cryptographic Voting — A Gentle Introduction.
[5] Katharina Bräunlich and Rüdiger Grimm. 2013. A Formal Model for the Requirement of Verifiability in Electronic Voting by Means of a Bulletin Board.
[6] Jonathan Katz and Yehuda Lindell. 2015. *Introduction to Modern Cryptography.* CRC Press.
[7] Steve Kremer, Mark Ryan, and Ben Smyth. 2010. Election Verifiability in Electronic Voting Protocols.
[8] Ben Smyth, Steven Frink, and Michael R Clarkson. 2017. Election Verifiability: Cryptographic Definitions and an Analysis of Helios and JCJ.