A Secure Voting Scheme based on Rational Self-Interest

Jayadev Misra*

May 21, 2012

Abstract

We propose a scheme for secure voting that involves the candidates themselves in implementing the voting system. It exploits the competing interests (rivalry) and mutual distrust among the candidates to force an honest election.

Keywords Secure voting, Rational self-interest, Absence of Coercion.

1 Introduction

This paper proposes a scheme for secure voting. Unlike existing schemes [1, 2], it involves the candidates themselves in implementing the voting system. It exploits the competing interests (rivalry) and mutual distrust among the candidates to force an honest election.

We consider a voting system in which a voter[1] is allowed to cast a single vote for (exactly one candidate). It is required to publish the number of votes cast for each candidate at the conclusion of voting. Our proposal, in its basic form is: the voter transmits her vote to all the candidates; the candidates store the votes and publish the list of votes at the end. This sounds outlandish. We modify this protocol by having a trusted agent who (1) encrypts each vote, (2) transmits the vote to each candidate, and (3) confirms to the voter that her vote has been recorded. The trusted agent is designed so that if it crashes or is compromised during polling: (1) no data about the votes cast will leak out, and (2) the election can continue by replacing the trusted agent by another trusted agent. These requirements prevent the trusted agent from simply counting the votes as they are cast. Instead, it relies on the candidates to keep the counts. The proposed protocol ensures that a dishonest candidate will not lose any vote that has been cast nor manufacture votes; it relies on the candidates to police each other for these guarantees. We first discuss manual voting at a precinct, and online voting in Section 4.2.

*This work is partially supported by National Science Foundation grant CCF-0811536.

[1]Henceforth, the voter is referred to in the feminine gender.

Manual Voting The voting procedure at a precinct is identical to the current practices from the voter's perspective. The voter is first authenticated by the precinct workers. Then she casts her ballot electronically in a booth. She is shown a confirmation of how she has voted on a display screen. If she agrees with what she is shown, she leaves the booth, otherwise, she may raise a dispute; we discuss dispute resolution below. The voter does not receive any physical evidence of how she has voted, nor does she have to monitor the outcome later to ensure that her vote has been counted appropriately. This prevents any form of coercion.

For each voting booth, the election commission supplies a network of communicating machines each of which has a different function. The voter casts her ballot on one machine. This machine communicates with other machines that record the vote, and yet other machines may display the result to the voter. At least two of the machines run the software supplied by rival candidates; they record each vote as a certificate signed by the other candidate's machine. Upon completion of balloting at the precinct, they publish the votes in such a way that the voters remain anonymous and the authenticity of the votes can be guaranteed. The contents of the lists published by different machines will be identical. The outcome of the election can then be computed from these lists by independent agencies.

A *session* corresponds to the casting of the vote by a single voter. A *player* during a session is either the voter or one of the machines. We use the term *machine* to denote a logical process. The machines may be implemented on a single physical machine, or as a network of physical machines. We assume that the machines and the software supplied by the election commission are fault-free, though no such assumption is made out about the software supplied by the candidates. See Section 5.1 for a discussion about software verification.

Disputes and their Resolution The machines that run candidates' software ("candidates' machines") may cheat in a variety of ways. A machine may show the voter that she has voted differently from her actual vote. Or, a machine may record a vote differently from what it shows the voter. Or a machine may change a vote in the final published list. Additionally, machines may collude in their cheating, for example, by agreeing beforehand to display the correct result to a voter and publishing identical, though different, results in their final lists.

Some forms of cheating will lead to disputes. For example, if the voter sees that her vote has been recorded differently from how she has voted, she may raise a dispute; or, she may not, if she is dishonest and believes that the result supports her candidate. Also, if the final published lists from various machines do not agree, the electorate can raise a dispute. There are other kinds of cheating, involving collusion, that may not raise any dispute.

To resolve disputes that may arise the network records the content and time of transmission of each message only for the present session. The recording is erased if the session ends without dispute. A dispute is resolved by consulting the log of transmitted messages and identifying the dishonest players. Disputes involving final published lists are resolved by mere inspection. If all players behave honestly, there is never any dispute.

The candidate corresponding to a guilty machine is punished severely, to the extent of losing the election or being subjected to other severe civil or criminal penalties. A voter found to be guilty is also severely punished. It is never in the interest of a player to be identified as being dishonest even though it may cheat.

Rational self-interest implies that a candidate machine never takes a step that may implicate it, even though the machine may be dishonest. We show that rational steps, even though dishonest, will force honest outcomes.

Coercion An ideal voting system should make it impossible for any voter to be coerced to vote in a certain way; nor can a voter provide any evidence of how she has voted to gain any monetary benefits. Consequently, the published lists of votes should not identify the voters. In fact, no information about the voting pattern should be discernible from the published lists, because if the list of votes show the sequence in which voting occurred, for instance, they leak information that can be exploited by a coercer (a boss might tell his workers to cast their votes first thing in the morning for a certain candidate).

We avoid coercion in two ways: (1) the voter is not given proof of her vote, thus eliminating any demands from a coercer for a proof, and (2) a published vote carries no information about the voter or the sequence in which votes were cast. We propose a new scheme to eliminate coercion in online voting; see Section 4.2.

Aspects addressed, and those ignored The proposed scheme addresses most of the requirements of an honest election: (1) authentication: only and all eligible voters are allowed to vote, (2) single vote: eligible voters may vote at most once, (3) correct recording: each vote is counted appropriately, and (4) absence of coercion: no one's vote is revealed, and no one may be coerced to vote in a certain way. Authentication requirement, (1), in manual voting is addressed manually; we exploit the rivalry among the precinct workers representing different candidates to keep authentication honest. We suggest a novel authentication scheme for online voting that eliminates coercion. Requirement (2) is met by the design of the machines that allows only one vote to be cast per voting session, but we have no new proposal to address this issue in online voting. This paper addresses the other two concerns, correct recording (3) and absence of coercion (4).

Like all voting schemes, ours has certain limitations. We assume that the hardware of the machines and the communication network are fault-free. We do not address side-channel attacks: a machine may transmit votes wirelessly to another device thus revealing the sequence of votes; a machine may take photographs of the voters and remember how they cast their votes so that they are denied privacy; a voter may be required by her boss to carry a camera and capture a screenshot that shows that a vote has been cast as instructed by the boss; or a precinct worker may deliberately load different software from the one supplied by a candidate. We have to rely on reasonable social practices to ensure that such attacks are eliminated. We address some of the engineering and social concerns in Section 5.

2 Voting Schemes

We first consider voting for a single office for which there are just two candidates, m and n. We relax these constraints in Section 4.1. We will describe the operation of a single voting booth that allows one voter to vote at a time.

Precinct workers, representing the candidates, authenticate each voter. Because of their conflicting interests, we assume that only and all eligible voters are authenticated. After authentication, the precinct workers together allow the voter to enter a voting booth. We consider two schemes, *Scheme A* and *Scheme B*, in this

section. In Scheme A, the machine supplied by the election commission generates a random number in a session, encrypts the vote and the generated random number with its own key, and transmits the message to the candidates' machines. In the Scheme B, the election commission machine merely generates random numbers; all other functionalities are delegated to candidates' machines.

2.1 Scheme A

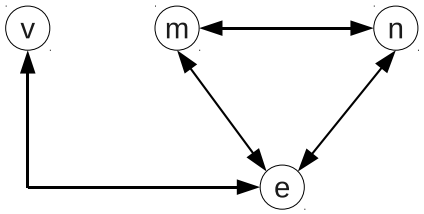

Figure 1: Voting Network for Scheme A

Honest Execution The communication network for this voting scheme, deployed in each voting booth, is shown in Figure 1. Node v represents the voter, e the machine supplied by the election commission, and m and n are the candidates' machines. The expected honest behaviors of the components are as follows. Voter v sends her choice of candidate x —where x is either m or n— to machine e. Machine e generates a random number i, and encrypts the pair (i, x) using its *own* encryption function; denote this encrypted value by $(i, x)_{\overline{e}}$. It sends this value to both m and n. On receiving $(i, x)_{\overline{e}}$ from e, each of m and n creates a digitally signed certificate from the received message; denote the certificate created by m by $((i, x)_{\overline{e}})_m$. Call i the *index* of this certificate and $(i, x)_{\overline{e}}$ its content; see [6] for a discussion of digital signatures. Certificates are assumed to be non-malleable, unforgeable, incorruptible and verifiable. Each machine sends its certificate to its rival. Upon receiving a certificate from its rival, a machine verifies that its content matches the value received from e. If it does, it sends an acknowledgment to e; if it does not, it raises a dispute. Machine e sends a confirmation to v after receiving acknowledgments from both m and n, and then ending the voting session.

Machines m and n store the certificates they receive from their rivals. On completion of balloting for the precinct, they both publish the list of certificates *sorted by their indices*; the sorting step obliterates the order in which the votes were cast. Then, the election commission publishes the decryption function for e. Note that e does not store the decryption function corresponding to its own encryption, to prevent leaking information if it is compromised. Now each certificate's content pair (i, x) can be recovered: first apply the encryption function of m to $((i, x)_{\overline{e}})_m$ to recover $(i, x)_{\overline{e}}$ and then the decryption function of e to this value to recover (i, x). Independent organizations can ensure that each certificate is valid and that the lists are identical in their contents. Then the number of ballots cast for each candidate is easily computed (from either published list).

Honesty Assumptions We assume that machine e, programmed by the election commission, carries out its steps honestly and that the hardware of the ma-

chines and the communication network are fault-free.

Machine e has a few simple tasks to perform: (1) generate random numbers, (2) encrypt the pair (i, x), and (3) receive and transmit messages. It does not store any information about the vote after a session is completed. Its program is so simple that it can be publicly scrutinized and formally verified.

Dishonest Execution Unlike machine e, Machines m and n may be dishonest. They may send inappropriate certificates, or no certificate at all. Further, they may create forged certificates or lose some of the certificates they have received, so that their published lists are incorrect. The first form of dishonesty is easily checked by the rival who raises a dispute. We show in Section 3.3 that publishing an incorrect list will immediately identify the dishonest player, who will then be subject to severe punishment.

The voter may cheat by claiming that the vote that has been cast differs from her intention. Again, this will be identified as dishonest and the voter punished; see below. Absence of output, given valid and timely inputs, is also regarded as dishonest, and the player to receive that output should raise a dispute, if it is honest.

Dispute Resolution A dispute arises during a voting session in the following situations.

1. Machine e never receives an expected message. The message could be from the voter, or an acknowledgment from m or n. After a suitable time period, the machine raises a dispute.
2. Machine m or n receives an inappropriate certificate from its rival. It then raises a dispute.
3. Voter v raises a dispute if she does not receive a timely confirmation from e.

We assume that each player has a mechanism to notify the authorities in case of a dispute. The authorities resolve a dispute by taking a log of the messages transmitted during a session between e, m and n. But they do not examine the contents of the channel from v to e that will reveal the vote cast. They check the machines m and n as follows: (1) recover the contents of the certificates they have sent to each other (by encrypting each by the corresponding machine's public key) and check if the contents match the value sent to by e, and (2) determine if a machine has sent an acknowledgment to e. These checks either identify one of the machines to be dishonest or exonerate both of them, in which case v is dishonest, because the remaining machine, e, is honest. Thus, dispute resolution does not reveal the vote; it only uses publicly available information about encryption functions.

No player can afford to be identified as being dishonest. So, there is no dispute in a rational execution. We prove this result formally for the Scheme B which includes Scheme A as a special case.

A dispute may arise after the publication of the lists if they are found to be non-identical. No list will contain a corrupt certificate because that will be immediately recognized and the culprit punished severely. We will show in Section 3.3 that neither m nor n will suppress a certificate if they behave rationally. Therefore, the published lists will contain the certificates corresponding exactly to the votes cast.

Need for Random Numbers, Encryption and Signatures Every vote has an index, a random number, to distinguish it from another vote cast in the same way. Without an index, a candidate can duplicate a certificate from its rival and claim that a certain vote has been cast by two voters. The votes are encrypted so that no candidate knows what vote has been cast, and the voter intention is not revealed during dispute resolution. This is especially important in online voting to prevent coercion; see Section 4.2. The votes have to be signed by the candidates to prevent duplication by a rival, and also to prevent a candidate from later disowning a vote.

2.2 Scheme B

In Scheme A of Section 2.1, the election commission supplies machine e that is deemed honest. Establishing the honesty of e requires verification of its software. Even though the program is simple, we can still avoid most of the verification by delegating its major functionalities to the candidates' machines. The resulting scheme has one major drawback; dispute resolution requires revealing the vote. Though a rational execution is dispute-free (see Proposition 8 in Section 3.2), this is a serious breach of voter privacy; it should be used only when the expectation of dispute is negligible or revealing how a particular voter has voted to the authorities is of no concern.

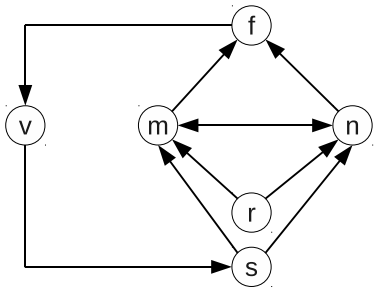

Figure 2: Voting Network for Scheme B

In the network of Figure 2, the functionalities of e reside in three different processes s, r and f. Process s reads input x from v and sends it to both m and n. Process r generates a random number for each voting session and sends it to both m and n. Machine m sends certificate $(i, x)_m$ to n and n sends $(i, x)_n$ to m. Each machine then validates the certificate it receives against i and x supplied by r and s, respectively. Then each of m and n sends an acknowledgment x to f. Process f, on receiving both acknowledgments, sends message x to v. Observe that the votes sent by s are in plain-text; so, the candidates' machines know how the vote is cast.

The assignment of software responsibilities to various parties is as follows: (1) the election commission is responsible for r, (2) the candidates are responsible for their corresponding machines, m and n, and (3) machines s and f are programmed by different candidates.

The only verification responsibility for the election commission is r. We note

that r has no input; the system detects a new voting session when the precinct workers authenticate the next voter. Then, all the message logs are initialized, and r sends the next random number.

Hardware random number generators are readily available [7]. Alternatively, we can use a publicly available cryptographically secure pseudo-random generator (CPRG). The program and its verification would be open to public scrutiny. The only hidden items will be the seed and the other parameters required for random number generation. For a CPRG, it is not possible to efficiently recreate the sequence given only the set of random numbers[2], a property essential to ensuring that no voting pattern in the sequence of votes cast be discernible from the published lists. We require that no number be generated twice during the voting at a precinct. Using a 40-bit key as the index, where each bit is randomly generated, in a run of 4000 (which corresponds to a large precinct), the probability of repeating a number is around 4×10^{-9}. If additional guarantee against repetition is required, a longer index can be used, or a test to ensure no number is being repeated, due to Floyd [3], may be employed.

Any player may raise a time-out dispute if it does not receive timely inputs. Additional disputes may arise during a session as follows.

1. Each of m and n raises a dispute if it receives an inappropriate certificate. Either the certificate is not properly signed, or its contents do not match random number i from r and/or vote x from s.
2. Process f raises a dispute if it receives acknowledgments that differ in their contents.
3. Voter v raises a dispute if it receives a message whose content contradicts its vote.

It is essential that s and f be programmed by different candidates. To see the necessity of this requirement, suppose s and f are both programmed by m. They may collude in the following way to change a vote for n to m. When v casts a vote for n, s instead sends vote m to both m and n, they exchange certificates, m sends n to f (and n sends m), and f sends n to v. The voter does not dispute, neither does n.

Dispute Resolution Dispute resolution in Scheme B is similar to that in Scheme A. As before, a log of the messages transmitted during a session is examined. Given the vote (the contents of the channel from v to s) and the random number generated by r (the contents of the channel from r to either m or n), the behaviors of all other players are determined. If any player has deviated from the expected behavior, either by sending an incorrect output or sending nothing at all, then it is guilty. The certificates sent by n and m are checked by first decrypting them with their public keys.

3 Correctness

Our goal in this section is to show that Scheme B of Section 2.2 is correct in the following sense: (1) every voting session ends without dispute, (2) the lists

[2]This observation is due to David Zuckerman.

published by m and n contain exactly the certificates for the votes that have been cast. Additionally, as should be obvious from the algorithm, the only information conveyed from the published lists is the number of votes. This implies that voter coercion is impossible because a voter carries no evidence from the polling booth of how she voted, nor can that information be deduced from the published lists (except if all votes are cast for a single candidate). Further, the published lists do not reveal any information about the times at which the votes were cast, nor about the voting patterns of any group of individuals, such as those who voted in sequence.

It is obvious that if every player is honest, the voting scheme meets its goal. Also, it is obvious that if the players are arbitrarily dishonest, the voting scheme fails. For example, if m and n collude to suppress a vote for n but report the correct vote to f, then the voting session ends without dispute; however, their published lists will not contain the votes as they were cast, thus violating a correctness requirement. We may address this problem by assuming that the candidates do not collude; in this case it is in n's interest to avoid collusion. But we do not need a strong axiom, such as absence of collusion, which may not even hold is some circumstances. We use a weaker axiom whose essence is that no player would dare to make the first dishonest move that would implicate it.

We define two machines to be *cohorts* if they have been programmed by the same candidate; machines that are not cohorts are *rivals*. The cohort relation is an equivalence relation among machines. The voter could be a cohort of some candidate, but all it can do to help is vote for the candidate. It can raise a dispute during its session in the hope of subverting the election, but a frivolous dispute will implicate it.

We can expect cohorts to collude by, possibly, communicating additional information in order to cheat the voting system and gain a competitive advantage for their candidate. But rivals have diametrically opposite interests. If a player can dispute and implicate a rival it will do so; therefore, a player should never take a step that will present a rival with such an opportunity. A player's step is irrational if it does present a rival with such an opportunity; it is rational otherwise. We show that rational behaviors, even though dishonest, lead to honest outcomes. We define these notions formally in the next section. We consider a single voting session and prove that it ends without dispute, and that each of m and n holds a certificate from the other for the vote that has been cast. In Section 3.3, we show that each candidate publishes exactly the set of certificates received.

3.1 Terminology and Notation

A *step* is a triple of the form (p, x, q) denoting that message x has been transmitted along channel (p, q) during a voting session. An *execution* of a voting session is a set of steps. An execution corresponding to a completed session is called a *trace*. Step (p, x, q) occurring in a trace denotes the sending of x by p as well as the receiving of x by q, since the channels are deemed fault-free and a trace is a completed execution.

A *predicate* is a step, or conjunction of steps and/or their negations. A predicate denotes the set of traces in which the given steps have occurred and their negations have not occurred. For example, $(s, m, n) \wedge (r, i, n) \wedge \neg(n, m, f)$ denotes the set of traces in each of which (s, m, n) and (r, i, n) occur and (n, m, f) does not.

Dispute Predicates The following predicates describe the conditions under which various players may legitimately raise disputes. For example, m can raise a dispute if it has received some x from s, some i from r, but not $(i, x)_n$ from n. In each case, the absence of timely input or appropriate input raises a dispute. Therefore, player r, that has no input from another player, never disputes. Below, x ranges over values from $\{m, n\}$ and i is any integer.

m: $(s, x, m) \wedge (r, i, m) \wedge \neg(n, (i, x)_n, m)$
n: $(s, x, n) \wedge (r, i, n) \wedge \neg(m, (i, x)_m, n)$
f: $(n, x, f) \wedge \neg(m, x, f)$ or $(m, x, f) \wedge \neg(n, x, f)$
v: $(v, x, s) \wedge \neg(f, x, v)$

A trace is *p-disputable* if the dispute predicate holds for p in that trace. Player p is *honest* in a trace if its outputs are correct given its inputs.

We assume that a player raises a dispute only if it will implicate a rival, not a cohort. For example, f does not raise a dispute if it deems that only its cohort n is dishonest. A p-disputable trace may have ended without dispute because p may not have raised a dispute even though it had the option to do so. The formal proof does not require a player to raise a dispute whenever it has the option of doing so.

The *state* of a player in a trace is the set of steps in which it has sent or received a message. Step (p, x, q), where p and q are rivals, is *irrational* in a given state of p if there is a trace t such that:

1. t includes p's given state and the step (p, x, q),
2. all rivals of p are honest in t, and
3. t is q-disputable.

Trace t, as described above, is called a *witness* to the irrationality of the step. A step that is not irrational is *rational*.

Rationality Axiom Players take only rational steps.

The justification for this axiom is as follows. Consider a step (p, x, q) that is irrational with trace t as a witness. Now, q can dispute in t and the dispute will implicate some cohort of p (because rivals of p are honest in t). This is a possible execution that is against the interest of p. Therefore, rationally, p should not take such a step.

An execution in which every player takes rational steps is a *rational execution*, and similarly we define *rational trace*. An honest execution is rational, but not conversely. A player may send dishonest messages to its cohorts in a rational execution. Our proof will show that such a step is useless and the receiving cohort will have to ignore it in order to take a rational step.

A player who receives valid inputs sends its output to a rival in timely manner. Waiting too long to output is irrational, because the rival could raise a dispute implicating this player. Henceforth, we assume that in every rational trace if a player has valid inputs, it sends some output to a rival.

In the following proofs, we prove the irrationality of a step by displaying a witness trace. In those cases where we can prove irrationality, we show that the only possible rational step is the honest step.

3.2 Correctness of a voting session

For predicates b and c, $b \Rightarrow c$ means that if b holds in a *rational trace*, then so does c. Most propositions we prove are of the form $b \Rightarrow c$ where b includes a step in which p receives a message, and c is a step in which it sends a message to rival q. Typically, we derive predicate b' from b by assuming that rivals of p are honest and using the results of other propositions. Then we show that $b' \wedge \neg c$ implies the dispute predicate of q; that is q can possibly dispute and its doing so will implicate a cohort of p. Therefore, taking any step other than c is irrational. Predicate b' denotes some honest trace, and, hence, there is some rational trace where b' holds.

Henceforth, we assume the following cohort relation without loss in generality: $\{s, m\}, \{n, f\}, \{v\}, \{r\}$. To see the main idea behind the proof, consider a step that machine n is about to take after receiving i from r and x from s. It cannot rationally send anything other than $(i, x)_n$ to m because: (1) it can not assert that its rivals have been dishonest; so, (2) m may immediately raise a dispute and implicate n if it receives something different from n. Next, consider m that has just received $(i, x)_n$ from n. It reasons exactly as above to deduce that n has received i from r and x from s, otherwise it would have sent something different to m. Therefore, if m sends anything other than $(i, x)_m$, n can raise a dispute implicating either m or s, a cohort. As the chain of message transmission grows, the reasoning gets more complicated. So, we have adopted this formalism to structure and simplify the proofs.

Proposition 1

$(s, x, n) \wedge (r, i, n) \Rightarrow (n, (i, x)_n, m)$, for all x.

Proof: Assume that the rivals of n are honest. Therefore, s and r are honest.

$$\begin{array}{rl} & (s, x, n) \wedge (r, i, n) \\ \Rightarrow & \{s \text{ is honest and } r \text{ is honest.}\} \\ & (s, x, m) \wedge (r, i, m) \end{array}$$

Now, $(s, x, m) \wedge (r, i, m) \wedge \neg(n, (i, x)_n, m)$ implies the dispute predicate for m.

Proposition 2

$(n, (i, x)_n, m) \Rightarrow (m, (i, x)_n, n)$, for all x.

Proof: Assume that the rivals of m are honest. Therefore, n is honest.

$$\begin{array}{rl} & (n, (i, x)_n, m) \\ \Rightarrow & \{n \text{ is honest}\} \\ & (s, x, n) \wedge (r, i, n) \end{array}$$

Now, $(s, x, n) \wedge (r, i, n) \wedge \neg(m, (i, x)_m, n)$ implies n's dispute predicate. □

Suppose s has been dishonest in the message it sent to m, by either sending nothing or a value different from what it sent to n. Player m, on receiving the message from n knows exactly what n has received and what s has sent to n. From this proposition, m has no choice but to to shield its cohort s by sending back the certificate corresponding to the vote in n's certificate.

Proposition 3

$(n, (i, x)_n, m) \Rightarrow (m, x, f)$, for all x.

Proof: Assume that the rivals of m are honest. Therefore, n and f are honest.

$(n, (i, x)_n, m)$

$\Rightarrow$ {from Proposition 2}

$(n, (i, x)_n, m) \wedge (m, (i, x)_n, n)$

$\Rightarrow$ {n is honest. So, $(n, (i, x)_n, m) \Rightarrow (s, x, n) \wedge (r, i, n)$}

$(s, x, n) \wedge (r, i, n) \wedge (m, (i, x)_n, n)$

$\Rightarrow$ {n is honest.}

(n, x, f)

Now, $(n, x, f) \wedge \neg(m, x, f)$ implies the dispute predicate for f.

Proposition 4

$(m, x, f) \Rightarrow (f, x, v)$, for all x.

Proof: Assume that the rivals of f are honest. Therefore, m and s are honest.

(m, x, f)

$\Rightarrow$ {m is honest.}

(s, x, m)

$\Rightarrow$ {s is honest.}

$(v, x, s) \wedge (s, x, n)$

Now, $(v, x, s) \wedge \neg(f, x, v)$ implies the dispute predicate for v.

Proposition 5

f does not raise a dispute if $(m, x, f) \wedge \neg(n, x, f)$ holds, for any x.

Proof: From the proof of Proposition 4, $(m, x, f) \Rightarrow (f, x, v)$. So, f does not raise a dispute. If $(m, x, f) \wedge \neg(n, x, f)$ holds, then n is dishonest, and f shields its cohort n in that case.

Proposition 6

$(v, x, s) \Rightarrow (s, x, n)$, for all x.

Proof: Assume that the rivals of s are honest. Suppose s sends y to n.

(s, y, n)

$\Rightarrow$ {r is honest and sends some index i to n.}

$(s, y, n) \wedge (r, i, n)$

$\Rightarrow$ {n is honest}

$(n, (i, y)_n, m)$

$\Rightarrow$ {from Proposition 3}

(m, y, f)

$\Rightarrow$ {from Proposition 4}

(f, y, v)

Thus, $(v, x, s) \wedge (s, y, n)$ implies $(v, x, s) \wedge (f, y, v)$ which implies the dispute predicate of v if $x \neq y$.

Proposition 7

Player v receives confirmation for the vote she had cast.

Proof:

(v, x, s)

$\Rightarrow$ {from Proposition 6}

(s, x, n)

$\Rightarrow$ {from Proposition 1}

$(n, (i, x)_n, m)$, for some i (*1)

$\Rightarrow$ {from Proposition 2}

$(n, (i, x)_n, m) \wedge (m, (i, x)_m, n)$ (*2)

$\Rightarrow$ {from Proposition 3}

(m, x, f) (*3)

$\Rightarrow$ {from Proposition 4}

(f, x, v) (*4)

Corollary 1 Players m and n receive valid certificates from their rivals for the vote that has been cast.

Proof: Follows from (*2) in the proof of Proposition 7.

Proposition 8

A rational trace is dispute-free.

Proof: We assert that no player raises a dispute. We need only consider the players $\{m, n, f, v\}$ that have dispute predicates. Suppose (v, x, s) holds for some x. Consider the proof of Proposition 7. Dispute predicate for m does not hold, from (*1). Dispute predicate for n does not hold, from (*2). Dispute predicate for f has two parts: (1) $(n, x, f) \wedge \neg(m, x, f)$ does not hold, from (*3), and (2) from Proposition 5, f does not raise a dispute if the other part of the dispute predicate, $(m, x, f) \wedge \neg(n, x, f)$, holds. Dispute predicate for v does not hold, from (*4).

3.3 Correctness of the published lists

We have established in Proposition 8 in Section 3.2 that every rational voting session is dispute-free, and in the Corollary of Proposition 7 that each of m and n receives certificates from its rival for every vote that is cast. Let S_m and S_n be the list of certificates received by m and n, in the sorted order by their contents. Upon completion of voting, they are required to publish these lists. They may be dishonest and publish different lists. We show that, using the Rationality Axiom, they must publish S_m and S_n.

One of m and n publishes its list first, say m (formally, the publication by m is not causally dependent on the publication by n; see [4]). Suppose m publishes a certificate that is not in S_m. Such a certificate would be seen to be invalid, because either it is not signed by n or it is a duplicate of one signed by n; in each case, m will be punished. Suppose m does not publish a certificate that is in S_m. We claim that this is irrational, because n can publish S_n that would contain the corresponding certificate, implicating m. Therefore, m publishes S_m. Now, n has to publish S_n; otherwise, it will be implicated.

4 Extensions

4.1 Multiple Candidates

We propose generalizations when there are more than two candidates. We can not then assume that all pairs of candidates are rivals. The schemes are applicable even when some of the candidates collude. All that is needed is that not all candidates collude together, i.e., there are at least two camps so that some pair of candidates are rivals.

Scheme A of Section 2.1 is generalized by having each candidate machine send its certificates to all machines. At the end of balloting, every machine publishes all certificates, grouped by different machines, in sorted order. If it is deemed infeasible to require the candidates to keep track of all other candidates, a candidate's machine sends its certificate to e who verifies it and sends it to all other machines.

Scheme B of Section 2.2 requires a more significant modification. The election commission can not choose programs for s and f from rival factions because (1) there may be several rival factions, and (2) it can not identify the rival factions. Instead, we require that s be programmed by the election commission. Each candidate machine sends certificates to all other candidate machines. The program for f is picked randomly from the ones submitted by all the candidates, in order to maintain symmetry among the candidates. It is not essential to assume that f is honest. All the propositions proved in Section 3.2 still hold if only s is assumed to be honest but f is arbitrary, because making component s more honest does not affect the correctness of the propositions.

4.2 Online Voting

The manual voting schemes described so far prevent a candidate's machine from communicating with any external machine (besides the ones explicitly shown) or retaining information about the voting pattern. In online voting, the machines may identify voting patterns, or a voter's authentication information may be demanded by a boss who can then cast the vote on her behalf. We propose a novel authentication protocol to address coercion and a method by which voting pattern information can be destroyed.

We propose to use Scheme A of Figure 1 (page 4) for online voting. As before, e is honest and the machines and hardware are fault-free. Current technology is mostly adequate for fault-free communication, particularly since the number of messages and the rate of message transmission are both very low.

A Novel Authentication Scheme A major problem in online voting is authentication, more specifically, authentication that prevents coercion. Imagine a boss demanding all the authentication information from a subordinate and then himself voting as the subordinate.

A typical authentication attempt has two possible outcomes: (1) it succeeds, which allows the voter to vote, or (2) it fails and the voter is not allowed to vote. We permit an authentication attempt to have three possible outcomes: (1) it succeeds, which allows the voter to vote, (2) it *nearly* succeeds, which allows the voter to vote; the voter is made to believe that she has voted successfully, but the vote is not counted, or (3) it fails and the voter is not allowed to vote.

Usually, an authentication requires the voter to provide a name and a "password" in addition to possible biometric data such as iris scan, finger print or voice print. For each genuine password we propose to have a set of "fake" passwords that are different from the genuine. For example, a fake password may differ in 5 symbols from the genuine one, or the fake and the genuine match only for a proper prefix. The rule for creating a fake password from a genuine will be publicly available. An authentication attempt nearly succeeds if it presents a fake password in addition to all other required data.

The voter can supply a fake password to the coercer, or login with a fake if the coercer is watching. The response to a vote with a fake password is exactly

the same as for a genuine password. However, the vote is not counted; see below for further discussions. Coercion is now meaningless, because a voter can not be coerced to vote for a particular candidate nor can she provide proof that she actually voted in a certain manner.

Fake Voting It is not sufficient for e to reject a vote when presented with a fake password. A candidate's machine may keep a tally of when the votes were cast and deduce information about whether a coercer succeeded. To overcome such problems, e actually sends a vote to all the machines, but *for a fake candidate*. The candidates' machines can not decipher the vote since it can only be decrypted by e. So, the normal voting pattern is preserved. The votes for the fake candidate in the published lists will be discarded. Further, if it is important to avoid deduction about the degree of coercion from the lists published after the election, or the timing of vote casting, even the fake votes could be faked, by e voting for the fake candidate with high frequency.

4.3 End-to-end verifiability

End-to-end verifiability of an election system guarantees that upon completion of the election every voter can check that her vote has been counted. The schemes proposed in this paper do not provide such guarantee. We propose a variation of Scheme A that allows a voter to verify that her vote has been counted, at the expense of violating one of our earlier requirements on e, namely that it stores no information across voting sessions.

The simple extension is to tell the voter the index generated on her behalf; she can then check the index against the published lists. This extension has serious consequence for coercion. A coercer may demand to know the index. The voter can not simply make up an arbitrary index, as she could for on-line authentication, because the coercer will check the index against the published lists. So, we propose a slightly more elaborate subterfuge. Scheme A will be extended to allow a voter to cast fake votes for one or more candidates. Such votes will be treated exactly like genuine votes, but machine e keeps a count of the number of fake votes cast for each candidate and publishes it separately after the publications of the voting lists, to allow computation of vote counts. The voter can show the index corresponding to a fake vote to her coercer. In order to avoid detection of any pattern of fake voting for specific candidates —say one candidate receives hundreds of fake votes whereas others have none— machine e itself casts fake votes for all candidates such that every candidate receives equal number of fake votes. Again, we assume that e's software and hardware are infallible.

Here we would like to contrast two different notions of verifiability, *static* and *dynamic*. Consider the software correctness problem. Given a program for, say, sorting a list of numbers, we may either prove the program correct for all inputs by supplying a static proof, or we may verify that the output is correct each time the program is run with some specific input, a dynamic proof. Every voter needs a proof that her vote will be counted. She can rely on a static proof, the kind we have advocated, by having the software be subject to public scrutiny and formal verification. Or, she may demand a dynamic proof when she votes so that she can later establish from the published lists that her vote has been counted. Static proofs are best employed when the conditions under which the software operates are tightly

controlled, as would typically be the case in running a sorting program. Dynamic proofs are essential in a more chaotic environment, where other aspects, such as side-channel attacks, are real. Dynamic proofs are often preferred in voting, not only because the environment is chaotic, but also for the social acceptability of electronic voting. We advocate static proofs because the suggested voting schemes are simple enough that automatic verification of the associated software is practical today.

5 Designing a Practical Voting System

We have advocated an unusual voting system that may seem impractical at the outset. In this section, we separate the technical and non-technical concerns in this voting system. We argue that that the technical concerns are completely solvable. The non-technical concerns, though unusual, seem no more difficult than the ones in the traditional schemes.

5.1 Technical Concerns

We have assumed in Section 2.1 that (1) the hardware of the machines and the network are error-free, and (2) the software provided by the election commission is error-free. We can meet these assumptions reasonably. First, let all the machines be implemented on a single physical machine, and the channels be implemented as words in the shared memory. Each machine runs in its own partition, so that it does not affect any other machine. A machine sends a message along a channel by writing into a specific word in the memory, and it can write into a word at most once during a session. A machine reads from a channel by reading the value of the shared word. The shared words are reset (to some specific value) at the beginning of a voting session. Currently available technology for fault-tolerance through replication should be adequate for fail-stop systems with independent faults. If the degree of replication is large enough, it would also tolerate Byzantine faults [5]. After each logical machine publishes the list of votes, the contents of the partitions are erased so that no trace of the computation remains in the machine.

We insist that the candidates provide the software but not the machines on which the software runs. This prevents a machine from recording additional information about votes which the candidates may later exploit, to gain information about voting patterns, for instance.

5.2 Non-technical Concerns

There are many side channel attacks that may compromise an election. There is no technical fix for such problems. Some of the issues in side-channel attacks can be resolved by adopting strict manual procedures. For example, a bipartisan committee could oversee the loading of candidates' software in the machines.

The most obvious non-technical problem is the requirement that the candidates provide the software. It is reasonable to argue that a candidate has no legal obligation to provide software. If a candidate contracts with another party to deliver the software, he may have no guarantee that the contractor has the same interest as the candidate, and, it is not clear who should be punished in case of a dispute. This

seems to be a social problem based on trust. Establishing trust in software through automatic verification is a viable approach.

The specification of candidates' software is simple and precise. We expect that software vendors will market software meeting the specification. They will have to convince their clients, the potential candidates, of the correctness of their code. They can do so by providing a completely automated proof of the software along with the code. The proofs may be checked automatically by standard proof checkers.

Free market provides other incentives for building trust. A software vendor may take a large amount of insurance against failure of its software. The insuring companies will clearly need proofs of correctness along the lines we have sketched.

Another possible approach is for a community to build open software that is checked in a variety of ways by a multitude of experts. No one makes any money in this process, but the cost of ensuring honest elections is then largely eliminated.

6 Concluding Remarks

Every voting scheme is beset by engineering, social and legal problems. Ours is no exception. Ensuring that the machines and communication channels are fault-free is an engineering problem. The problem is essentially solved for closed systems, as in a polling booth. It is not so easily solved for communication over the internet. The non-technical problems, particularly, side-channel attacks, require different strategies for solution.

In spite of these shortcomings, we expect that voters will welcome a system that deviates very little from how they already vote, that is open to public scrutiny and yet provides the guarantees for an honest election.

Acknowledgment I am thankful to David Dill, Nickolai Zeldovich, Adam Klivans and Gordon Novak for discussions and advice on an earlier draft of this paper. Special thanks to David Zuckerman for discussions about cryptographically secure pseudo-random number generators. Mike Fischer, David Jefferson, Vladimir Lifschitz and Elaine Rich have provided many perceptive comments. Several members of IFIPS WG 2.3, in particular, Carroll Morgan, Ernie Cohen, Rajeev Joshi and Rustan Leino, were most helpful with their constructive criticism. Suggestions from the anonymous referees have improved this paper substantially.

References

[1] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[2] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *In USENIX Security Symposium*, pages 339–353, 2002.

[3] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, third edition, 1997.

[4] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[5] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *TOPLAS*, 4(3):382–401, July 1982.

[6] R.L. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb 1978.

[7] G. Taylor and G. Cox. Digital randomness. *Spectrum, IEEE*, 48(9):32 –58, september 2011.