Electronic Elections: Trust Through Engineering

Carsten Schürmann
IT University of Copenhagen
Copenhagen, Denmark
carsten@itu.dk

***Abstract*—Electronic voting technology is a two edged sword. It comes with many risks but brings also many benefits. Instead of flat out rejecting the technology as uncontrollably dangerous, we advocate in this paper a different technological angle that renders electronic elections trustworthy beyond the usual levels of doubt. We exploit the trust that voters currently have into the democratic process and model our techniques around that observation accordingly. In particular, we propose a technique of *trace emitting computations* to record the individual steps of an electronic voting machine for a posteriori validation on an acceptably small trusted computing base. Our technology enables us to prove that an electronic elections preserves the voter's intent, assuming that the voting machine and the trace verifier are independent.**

I. Introduction

The word *electronic* in an electronic election refers to both, the electronic method of casting a vote and the electronic method of computing the final result. Any specification of requirements for an electronic election needs to span the entire process, from voter registration to the approval of the final election result. Considering that our understanding of trust, safety, secrecy, availability, privacy, and correctness in terms of logic and formal specification is still at large implies that traditional software verification techniques currently do not (yet) apply to this setting.

In this paper we therefore leave the idea of formal verification and mathematical specification behind, and focus instead on the engineering aspects of as electronic election system by ensuring that the components that are responsible for gaining trust into the traditional voting process are mirrored by tangible components in the electronic counterpart. In particular, we believe that the following three principles justify to a significant degree why people believe in the electoral process and trust that the traditional election really measures the voter's intent.

1) The election process is *closed*, which means that no information other than the result may leak from the process. It follows that ballots must be *anonymized*, which is achieved, for example, by having them cast into a ballot box.
2) Representatives of the invested parties oversee the counting process. By there mere presence they introduce *accountability* into the process. It is not the voter who checks that a vote was counted correctly, this task is delegated to a representative.
3) In the case of dispute, there is the possibility of *recounting* the original and untouched physical evidence. This possibility constitutes a safety net and contributes to the trustworthiness of the process.

Therefore, we stipulate that an electronic election system must be held *accountable* for the result of an election. Its design must therefore be *transparent*, which means that it should be possible to follow and verify the steps of a computation, and for all practical purposes, this can only be achieved if the voting machine provides *abstract*, *anonymous*, and *tangible* evidence. Our design is inspired by the frog model [SBR01], as we make a difference between the *voting machine* (vote generation machine), and the *trace machine* (vote casting machine). That we reduce the trusted computing base from some a "standard interface that performs a very simple set of functions" to a small, easy to implement, and verified type checker is one of the main contributions of this work.

Principle 1: We implement the voting generation machine as an *abstract machine* on an untrusted computing platform. The physical machine simulates the individual steps of the voting machine algorithm and records the trace. Consequently, it has access to the memory locations where the counters are stored. In this work, we focus on detecting unjustified updates to these counters without making ballots public. Privacy, security, and availability concerns are mostly orthogonal to the design of the abstract machine, and can be tackled using cryptographic methods, such as homomorphic encryption. Differently from the frog model, our voting machine does not only generate frogs but counts itself.

Principle 2: We propose that the voter (as usual) goes into the voting booth, casts the vote, obtains physical evidence (such as a frog) and casts it into the ballot box. This evidence contains the *operational meaning* of the ballot, i.e. the particular part of the computation trace that witnesses the counting of the ballot. This evidence may or may not be readable by the voter. In the later case, we permit scanning devices "representing" the different parties into the voting booth. A voter should only use the scanner that he or she trusts to interpret the content of the ballot and not to

misuse the information contained within. This way, even a configuration problems in the graphical user interface can be detected.

Principle 3: Technically, our solution employs *trace emitting computation* (TEC). The idea of TEC is orthogonal but closely related to the idea of *proof carrying code* (PCC) [Nec97] and which works as follows: while the electronic voting machine is running, it constructs a witness trace that accounts for every step it took. We employ logical framework [HHP93] technology to specify these operations and their respective meaning. Just as in the original proof carrying code work, we use a small trusted implementation of the LF type checker to certify traces and check them for equivalence. Therefore, we will be able to detect, if malware [CHF07] has compromised the functional behavior (but neither privacy nor secrecy) of the machine. The traces can be used in a recount.

In this work, we discuss the design of one single voting machine. We hope, however, that the reader can extrapolate an idea from reading this paper, how trace emitting computations could scale to the entire electoral process. For example, we conjecture that this technique can be used to verify the traces of algorithms that map voting totals into number of seats. Although further experimental work is necessary, we believe that this is a novel idea.

The paper is organized as follows. In Section II we define what we mean by voter's intent, followed by Section III where we outline the technique of trace emitting computations. We move then on and describe a sample election in Section IV, its encoding in the logical framework LF in Section V, and give an experience report on the ITU election in Section VI. Eventually we analyze our technique in Section VII discuss how the ideas scale to larger elections in Section VIII and conclude with Section IX.

II. Preliminary Definitions

We shall write V for the set of voters, B for the set of valid ballots, O for the set of possible outcomes of an election, and $+$ for the tallying operation. $+$ redefines the usual understanding of $+$, as we use it to add up ballots.

Definition 2.1 (Voter's intent): Let V be a set of voters who cast a vote, B the set of valid ballots, then $I : V \to B$ is called the voters intent.

We measure the voters intent as $\sum_{v \in V} I(v)$. Our interest focuses on the mathematical meaning of the voter's intent. It does not take any other factors into account that may mislead the voter, such as for example the user interface [Eve07]. Next we shift our attention towards the domain specific language used to program electronic voting machines that run on the abstract machine (simulated by the physical machine). From here on forth we refer to programs as expressions e that return (when executed) values for which we write w. The operational semantics relates expressions e and w for which we define in Section IV the judgment $e \hookrightarrow w$ (pronounced e evaluates to w). We write $\ulcorner b \urcorner$ for the value w that represents $b \in B$ in the domain specific language. We furthermore assume that the domain specific language provides (besides functional abstraction) an application operation, for which we write $\mathbf{apply}$. For simplicity, we assume that this domain specific language supports also lists $[\ldots]$, and the corresponding case construct. Let *dre* be the electronic voting program (to be defined in Section IV) that is bound to be executed on the electronic voting machine. Let there be n voters $V = \{v_1 \ldots v_n\}$ participating in the election.

Definition 2.2 (Trustworthiness): We call a program dre *trustworthy* if it can be guaranteed that

$$\textbf{apply } dre\ [\ulcorner I(v_1) \urcorner, \ldots, \ulcorner I(v_n) \urcorner] \hookrightarrow w$$

and $w = \ulcorner \sum_{v \in V} I(v) \urcorner$.

The central idea of this paper is to validate each individual run of *dre* against the voter's intent as opposed to for example public access to partial vote information [RS07]. The problem with proposals of this kind is, however, that they violate the first and last principle outlined in the introduction and therefore endanger trust.

III. Trace Emitting Computations

Trace emitting computations (TEC) are designed to record a particular run of an electronic voting machine. It creates a trace of all steps executed and all decisions made, in such a way that the trace can be used to redo the computation. Once a particular computation ends, the resulting trace may be inspected with respect to an a priori specified soundness policy. Any violation, for example, inconsistencies in the computation, can be recognized a posteriori. Checking a trace for soundness is synonymous to validating the result of the computation.

TEC can be seen as a dual to proof carrying code (PCC) [Nec97]. In PCC, one agrees a priori on a safety policy for code, which means under which conditions a piece of code is considered safe. In TEC, one agrees a priori on a soundness policy for code, which means under which conditions a trace is to be considered valid. In PCC the the proof is being checked before execution, in TEC the trace is being checked after execution. In PCC the proof describes a static specification of a piece of code, in TEC the trace describes the dynamic behavior of a piece of code. The short comings of TEC are also dual to PCC. In TEC the soundness policy assumes a model of execution that only approximates the a real machine, whereas in PCC, the safety policy assumes a model of safety that only approximates full correctness. With TEC we can gain trust into the dynamic behavior of code, in PCC we can gain trust into the static description of code.

In the remainder of this section, we make the idea of trace emitting computations precise and show that with a bit of clever engineering the result of an election can be

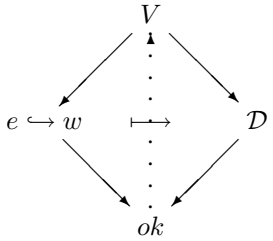

Figure 1. Diamond of trust

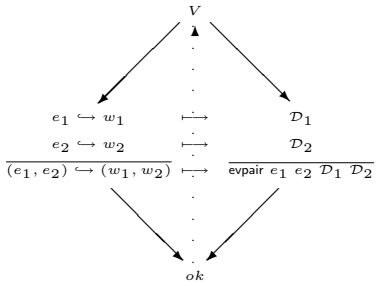

Figure 2. Compositionality

trusted beyond a reasonable level of doubt. Trust is created by separating the machine that computes, which we will simple refer to as the *voting machine* from another machine that administers the trace, which we will refer to as the *trace machine*. It is important, that those two machines are assumed independent. That is they do not share any common knowledge prior to an election that could be exploited to synchronize the two machines differently from the way that we describe below.

While the voting machine computes w from e it implicitly creates a trace $\mathcal{D}$. This trace is not directly shared with the voting machine and it will not be recomputed on the trace machine either; it will simply be reconstructed from the ballots. In TEC, ballots no longer contain only bits of information about which candidate was selected, they contain instead fragments of the trace of what the electronic voting machine did in order to count a vote. Therefore, trust is created by the voting machine channeling the trace fragment responsible for counting the vote through the voter in form of physical evidence, such as a printed ballot (which we will discuss in Section VI, depicted in Figure 6). The authenticity and correctness of the ballot is checked by the trace machine as well.

This idea is depicted in the diamond of trust in Figure 1. The top of the diamond depicts a voter V who would like to be convinced that if e evaluates to w, w is indeed the correct value. The solid arrows depict actions, for example the the arrow from V to $e \hookrightarrow w$ depicts the interaction of the voter with the voting machine. The arrow between V and $\mathcal{D}$ depicts the interaction between the voter and the trace machine. The arrow between $e \hookrightarrow w$ and $\mathcal{D}$ indicates the trusted communication of a trace (fragment) encoded in form of a ballot between the voting and the trace machine. The physical evidence must be carefully engineered; it must guarantee the basic democratic principles of a free and secret vote. For example, it should not be possible to sort the different trace fragments revealing information the order in which votes were cast.

The trust inducing scanners that are present in the voting booth pose an improvement over the traditional way elections are held. Since the rules of the election are fixed ahead of time and are public, the scanners may even display information about which candidate was not elected. This information could be extremely helpful to detect faults and absent names from the roaster displayed on the voting machine. At the end of the election day, the election officials collect the different trace fragments of $\mathcal{D}$ from the ballot box and attempt to verify that $\mathcal{D}$ is indeed a valid computation trace of a computation ending in $w = \ulcorner \sum_{v \in V} I(v) \urcorner$. Should the voting machine and the trace machine agree, we are confident to claim that the result can be trusted. If not, an irregularity was detected. Irregularities of this kind might occur, if a voter fails to follow instructions and does not cast the vote. This problem is well known, and occurs with the Belgian Jites and Digivote systems, and with the U.S. Populex system. What to do in this case needs to be determined by policy.

Note, that trust diamonds are compositional. Consider two separate computations $e_1 \hookrightarrow w_1$ and $e_2 \hookrightarrow w_2$, which the voting machine wants to pair together. By the rules of evaluation, $(e_1, e_2) \hookrightarrow (w_1, w_2)$. Figure 2 provides some evidence why we can trust the pairing operation. From the first and the second premiss we gain access to the traces $\mathcal{D}_1$ and $\mathcal{D}_2$ for validation, the instruction evpair combines traces to a new trace. If evpair can only be applied one way and the definition of evpair has been inspected ahead of time, and no other information is crossing the dotted arrow, we state with confidence that the evaluation of (e_1, e_2) is still trustworthy.

Under the assumption that the voting machine and the trace machine are truly independent, we claim that the election result deserves the trust of the people even though it was achieved without any human observer. Thanks to TEC, the reconstructed $\mathcal{D}$ contains all information, all decisions that the election machine did, and that can be verified by anyone post election. Although the individual cannot see his or her vote a posteriori, the trust into the system can be derived from the same principle as in a traditional election, namely that every step is explicit, and analyzed by all

invested entities.

IV. Sample Election

The rules that govern an election should be seen as a way to fill the diamond of trust with meaning. To declare the judgment, we shrink the form of the diamond of trust to $e \hookrightarrow w \diamondsuit \mathcal{D}$, whose meaning is defined below. First, we define the domain specific language for programming the election machine in terms of values w, expressions e, where we assume for simplicity, that there are only two different ballots A and B, and traces $\mathcal{D}$.

$$
\begin{array}{rcl}
w & ::= & x \mid \mathbf{A}* \mid \mathbf{B}* \mid \mathbf{lam}* \, x.\, e \mid \mathbf{0}* \mid \mathbf{s}* \, w \\
 & & \mid \mathbf{pair}* \, w_1 \, w_2 \mid \mathbf{nil}* \mid \mathbf{cons}* \, w_1 \, w_2 \\
e & ::= & !w \mid \mathbf{A} \mid \mathbf{B} \mid (\mathbf{case} \, x \, \mathbf{of} \, \mathbf{A} \Rightarrow e_1 \mid \mathbf{B} \Rightarrow e_2) \\
 & & \mid \mathbf{lam} \, x.\, e \mid \mathbf{apply} \, e_1 \, e_2 \mid \mathbf{0} \mid \mathbf{s} \, e \\
 & & \mid \mathbf{pair} \, e_1 \, e_2 \mid \mathbf{fst} \, e \mid \mathbf{snd} \, e \\
 & & \mid \mathbf{nil} \mid \mathbf{cons} \, e_1 \, e_2 \mid \mathbf{fold} \, e_1 \, e_2 \, (x \Rightarrow y \Rightarrow e_3) \\
\mathcal{D} & ::= & \mathsf{ev!} \mid \mathsf{ax}_A \mid \mathsf{ax}_B \mid \mathsf{caseA} \, e \, e_1 \, e_2 \;\; w \, \mathcal{D}_1 \, \mathcal{D}_2 \\
 & & \mid \mathsf{caseB} \, e \, e_1 \, e_2 \;\; w \, \mathcal{D}_1 \, \mathcal{D}_2 \mid \mathsf{evlam} \, (\lambda x.\, e) \\
 & & \mid \mathsf{evapp} \, e_1 \, e_2 \, (\lambda x.\, e_1') \, w_2 \, w \, \mathcal{D}_1 \, \mathcal{D}_2 \, \mathcal{D}_3 \\
 & & \mid \mathsf{evsucc} \, e \, w \, \mathcal{D} \mid \mathsf{evpair} \, e_1 \, e_2 \, w_1 \, w_2 \, \mathcal{D}_1 \, \mathcal{D}_2 \\
 & & \mid \mathsf{evfst} \, e \, w_1 \, w_2 \, \mathcal{D} \mid \mathsf{evsnd} \, e \, w_1 \, w_2 \, \mathcal{D} \\
 & & \mid \mathsf{evnil} \mid \mathsf{evcons} \, e_1 \, e_2 \, w_1 \, w_2 \, \mathcal{D}_1 \, \mathcal{D}_2 \\
 & & \mid \mathsf{evfold}_1 \, e_1 \, e_2 \, (\lambda x.\, \lambda y.\, e_3) \, w \, \mathcal{D}_1 \, \mathcal{D}_2 \\
 & & \mid \mathsf{evfold}_2 \, e_1 \, e_2 \, (\lambda x.\, \lambda y.\, e_3) \, w \, w_{11} \, w_{12} \, w_2 \, \mathcal{D}_1 \, \mathcal{D}_2 \, \mathcal{D}_3
\end{array}
$$

This language defines constants representing votes (here $\mathbf{A}$ and $\mathbf{B}$), functions, unary numbers, pairs and lists. We do not not define a type system, or a typing discipline. The computation model and the following program dre are known before the election, so the election program can be type checked ahead of time.

Example 4.1 (Voting machine code): We consider the following program that we would like to run on the voting machine.

$$
\begin{array}{rl}
dre = & \mathbf{lam} \, x. \, \mathbf{fold} \, x \, (\mathbf{pair} \, \mathbf{0} \, \mathbf{0}) \, (h \Rightarrow r \Rightarrow \\
 & \quad \mathbf{case} \, h \, \mathbf{of} \, \mathbf{A} \Rightarrow \mathbf{pair} \, (\mathbf{s} \, (\mathbf{fst} \, r)) \, (\mathbf{snd} \, r) \\
 & \qquad \mid \mathbf{B} \Rightarrow \mathbf{pair} \, (\mathbf{fst} \, r) \, (\mathbf{s} \, (\mathbf{snd} \, r)))
\end{array}
$$

Figure 3 defines the meaning of the evaluation judgment that emits the trace $e \hookrightarrow w \diamondsuit \mathcal{D}$. The rules that provide the respective evidence ax_A and ax_B witness that a vote has been cast. The subsequent two rules for caseA and caseB define the corresponding elimination forms, which allows a voting algorithm to case on which vote was cast.

In general for arbitrary elections there are usually more than only finitely many valid ballots. In more complicated scenarios where ballots require the voter to distribute percentages to the different candidates, there may even be infinitely many valid ballots. This is were our technique excels, in part because ballots are usually finitely axiomatizable.

The two rules for evlam and evapp allow the voting machine to work with functions. They describe the respective introduction and elimination laws. The rules for evzero and evsucc witness the counting, and the rules for evpair, evfst, and evsnd provide the possibility to work with n-ary counters used for aggregation toward the final result whereas the final four rules for evnil, evcons, evfold_1, and evfold_2 introduce introduction and elimination forms for lists.

This domain specific language for defining the software of a voting machine is based on a unary encoding of numbers. For a real electronic voting architecture we recommend working with a base-10 encoding, which would be best because humans can immediately read out the result of an election by inspecting the value.

The syntactic form of traces $\mathcal{D}$ are full of redundant information. Thus we take the freedom, and omit all inferable arguments from $\mathcal{D}$ when used in the remainder of the paper.

Next we illustrate the idea of trace emitting computations using a concrete example. Consider a situation where three voters cast a vote for two candidates in the order $\mathbf{A}, \mathbf{B}, \mathbf{A}$. The voters intent is therefore $\sum_{v \in V} I(v) = (2, 1)$. What we need to convince ourselves is therefore that

$$
\begin{array}{l}
\mathbf{apply} \; \mathit{dre} \; (\mathbf{cons} \, \mathbf{A} \; (\mathbf{cons} \, \mathbf{B} \; (\mathbf{cons} \, \mathbf{A}; \mathbf{nil}))) \\
\hookrightarrow \mathbf{pair} \; (\mathbf{s} \; (\mathbf{s} \, \mathbf{0})) \; (\mathbf{s} \, \mathbf{0}) \diamondsuit \mathcal{D}_0
\end{array}
\tag{1}
$$

The $\mathcal{D}_0$ can be more easily constructed if we introduce the following abbreviations.

$$
\begin{array}{rcl}
\mathit{empty} & = & (\mathbf{pair} \, \mathbf{0} \, \mathbf{0}) \\
\mathit{count} \; h \; r & = & \mathbf{case} \; (!h) \; (\mathbf{pair} \; (\mathbf{s} \; (\mathbf{fst} \; (!r))) \; (\mathbf{snd} \; (!r))) \\
 & & (\mathbf{pair} \; (\mathbf{fst} \; (!r)) (\mathbf{s} \; (\mathbf{snd} \; (!r))))) \\
\mathit{init} & = & \mathsf{evfold1 \; ev! \; (evpair \; evzero \; evzero)} \\
\mathit{votea} \; \mathcal{D} & = & \mathsf{evfold2 \; ev! \; } \mathcal{D} \\
 & & \quad \mathsf{(evcasea \; ev! \; (evpair \; (evsucc \; (evfstev!))} \\
 & & \qquad \mathsf{(evsnd \; ev!)))} \\
\mathit{voteb} \; \mathcal{D} & = & \mathsf{evfold2 \; ev! \; } \mathcal{D} \\
 & & \quad \mathsf{(evcaseb \; ev! \; (evpair \; (evfstev!) \; (evsucc} \\
 & & \qquad \mathsf{(evsnd \; ev!))))}
\end{array}
$$

Here, votea and voteb define the operational meaning a ballot for $\mathbf{A}$ and $\mathbf{B}$, respectively, which is independent of the current counter as it is λ-abstracted away. In more complex elections with, for example, fractional ballots there are many more (potentially infinitely many) valid ballots, where the operational meaning of a ballot becomes more complex and even more important. The following invariants hold.

$$
\frac{}{(\mathbf{fold} \; (!\mathbf{nil}*) \; \mathit{empty} \; \mathit{count}) \hookrightarrow (\mathbf{pair}* \; \mathbf{0}* \; \mathbf{0}*) \diamondsuit \mathit{init}}
$$

$$
\frac{(\mathbf{fold} \; (!L) \; \mathit{empty} \; \mathit{count}) \hookrightarrow (\mathbf{pair}* \; X \; Y) \diamondsuit \mathcal{D}}{\begin{array}{l}(\mathbf{fold} \; (!(\mathbf{cons}* \; \mathbf{A}* \; (!L))) \; \mathit{empty} \; \mathit{count}) \\ \hookrightarrow (\mathbf{pair}* \; (\mathbf{s}* \; X) \; Y) \diamondsuit \mathit{votea} \; \mathcal{D}\end{array}}
$$

$$
\frac{(\mathbf{fold} \; (!L) \; \mathit{empty} \; \mathit{count}) \hookrightarrow (\mathbf{pair}* \; X \; Y) \diamondsuit \mathcal{D}}{\begin{array}{l}(\mathbf{fold} \; (!(\mathbf{cons}* \; \mathbf{B}* \; (!L))) \; \mathit{empty} \; \mathit{count}) \\ \hookrightarrow (\mathbf{pair}* \; X \; (\mathbf{s}* \; Y)) \diamondsuit \mathit{voteb} \; \mathcal{D}\end{array}}
$$

$$\frac{}{A \hookrightarrow A \Diamond \mathsf{ax}_A} \qquad \frac{}{B \hookrightarrow B \Diamond \mathsf{ax}_B}$$

$$\frac{e \hookrightarrow A \Diamond \mathcal{D}_1 \quad e_1 \hookrightarrow w \Diamond \mathcal{D}_2}{\mathbf{case}\ e\ \mathbf{of}\ \mathbf{A} \Rightarrow e_1 \mid \mathbf{B} \Rightarrow e_2 \hookrightarrow w \Diamond \mathsf{caseA}\ e\ e_1\ e_2\ \ w\ \mathcal{D}_1\ \mathcal{D}_2} \qquad \frac{e \hookrightarrow B \Diamond \mathcal{D}_1 \quad e_2 \hookrightarrow w \Diamond \mathcal{D}_2}{\mathbf{case}\ e\ \mathbf{of}\ \mathbf{A} \Rightarrow e_1 \mid \mathbf{B} \Rightarrow e_2 \hookrightarrow w \Diamond \mathsf{caseB}\ e\ e_1\ e_2\ w\ \mathcal{D}_1\ \mathcal{D}_2}$$

$$\frac{}{!w \hookrightarrow w \Diamond \mathsf{ev!}} \qquad \frac{}{\mathbf{lam}\ x.\, e \hookrightarrow \mathbf{lam}\ x.\, e \Diamond \mathsf{evlam}\ (\lambda x.\, e)} \qquad \frac{\begin{array}{c} e_1 \hookrightarrow \mathbf{lam}\ x.\, e_1' \Diamond \mathcal{D}_1 \\ e_2 \hookrightarrow w_2 \Diamond \mathcal{D}_2 \\ [w_2/x]e_1' \hookrightarrow w \Diamond \mathcal{D}_3 \end{array}}{\mathbf{apply}\ e_1\ e_2 \hookrightarrow w \Diamond \mathsf{evapp}\ e_1\ e_2\ (\lambda x.\, e_1')\ w_2\ w\ \mathcal{D}_1\ \mathcal{D}_2\ \mathcal{D}_3}$$

$$\frac{}{\mathbf{0} \hookrightarrow \mathbf{0} \Diamond \mathsf{evzero}} \qquad \frac{e \hookrightarrow w \Diamond \mathcal{D}}{\mathbf{s}\ e \hookrightarrow \mathbf{s}\ w \Diamond \mathsf{evsucc}\ e\ w\ \mathcal{D}}$$

$$\frac{e_1 \hookrightarrow w_1 \Diamond \mathcal{D}_1 \quad e_2 \hookrightarrow w_2 \Diamond \mathcal{D}_2}{\mathbf{pair}\ e_1\ e_2 \hookrightarrow \mathbf{pair}\ w_1\ w_2 \Diamond \mathsf{evpair}\ e_1\ e_2\ w_1\ w_2\ \mathcal{D}_1\ \mathcal{D}_2}$$

$$\frac{e \hookrightarrow \mathbf{pair}\ w_1\ w_2 \Diamond \mathcal{D}}{\mathbf{fst}\ e \hookrightarrow w_1 \Diamond \mathsf{evfst}\ e\ w_1\ w_2\ \mathcal{D}} \qquad \frac{e \hookrightarrow \mathbf{pair}\ w_1\ w_2 \Diamond \mathcal{D}}{\mathbf{snd}\ e \hookrightarrow w_2 \Diamond \mathsf{evsnd}\ e\ w_1\ w_2\ \mathcal{D}}$$

$$\frac{}{\mathbf{nil} \hookrightarrow \mathbf{nil} \Diamond \mathsf{evnil}} \qquad \frac{e_1 \hookrightarrow w_1 \Diamond \mathcal{D}_1 \quad e_2 \hookrightarrow w_2 \Diamond \mathcal{D}_2}{\mathbf{cons}\ e_1\ e_2 \hookrightarrow \mathbf{cons}\ w_1\ w_2 \Diamond \mathsf{evcons}\ e_1\ e_2\ w_1\ w_2\ \mathcal{D}_1\ \mathcal{D}_2}$$

$$\frac{e_1 \hookrightarrow \mathbf{nil} \Diamond \mathcal{D}_1 \quad e_2 \hookrightarrow w \Diamond \mathcal{D}_2}{\mathbf{fold}\ e_1\ e_2\ e_3 \hookrightarrow w \Diamond \mathsf{evfold}_1\ e_1\ e_2\ (\lambda x.\, \lambda y.\, e_3)\ w\ \mathcal{D}_1\ \mathcal{D}_2} \qquad \frac{\begin{array}{c} e_1 \hookrightarrow \mathbf{cons}\ w_{11}\ w_{12} \Diamond \mathcal{D}_1 \\ \mathbf{fold}\ w_{12}\ e_2\ e_3 \hookrightarrow w_2 \Diamond \mathcal{D}_2 \\ [w_{11}/x][w_2/y]e_3 \hookrightarrow w \Diamond \mathcal{D}_3 \end{array}}{\begin{array}{l} \mathbf{fold}\ e_1\ e_2\ (x \Rightarrow y \Rightarrow e_3) \hookrightarrow w \Diamond \\ \quad \mathsf{evfold}_2\ e_1\ e_2\ (\lambda x.\, \lambda y.\, e_3)\ w\ w_{11}\ w_{12}\ w_2\ \mathcal{D}_1\ \mathcal{D}_2\ \mathcal{D}_3 \end{array}}$$

Figure 3. Diamond judgment, Arithmetic

Using these abbreviations, and a little bit of work, we define

$$\mathcal{D}_0 = \mathsf{evapp}\ \mathsf{evlam}\ \mathsf{ev!}\ (votea\ (voteb\ (votea\ init)))$$

and check that it is indeed a valid trace. This is the trace that we would expect the voting machine to construct according to (1).

At closing time, the voting machine will announce the outcome of the election is $\mathbf{pair}\ (\mathbf{s}\ (\mathbf{s}\ \mathbf{0}))\ (\mathbf{s}\ \mathbf{0})$. The ballot box will contain two pieces of physical evidence that contain $votea$ and one that contains $voteb$. Those ballots will be scanned (by a machine that is independent of the voting machine), the actual evidence of the election $\mathcal{D}_0$ constructed. Thankfully, no matter in which order the evidence is reconstructed, the election result is verified by checking the evidence against (1). As voting machine and trace machine are assumed independent, the technology will detect if votes have been tempered with. Conversely, the machine can also determine, if a voter did vote electronically, but did not cast the ballot into the ballot box.

Recall that in this paper we are concerned about the fact if a voting machine captures the voter's intent, which means that we are interested in detecting irregularities that occurred while counting votes. The study of recovering measures is left to future work.

Theorem 4.2: If

$$L = \mathbf{cons}{*}\ \ulcorner I(v_1) \urcorner\ \ldots\ (\mathbf{cons}{*}\ \ulcorner I(v_n) \urcorner\ \mathbf{nil})$$

and

$$\mathcal{D}_i = \begin{cases} votea & \text{if } I(v_i) = \mathbf{A} \\ voteb & \text{if } I(v_i) = \mathbf{B} \end{cases}$$

then $\mathbf{fold}\ (!L)\ empty\ count) \hookrightarrow w \Diamond \mathcal{D}_L$ iff $w = \ulcorner \sum_{i=1\ldots n} I(v_i) \urcorner$ and $\mathcal{D}_L = \mathcal{D}_1\ (\ldots (\mathcal{D}_n\ init))$.

Proof: by induction on n. In each case, apply one of the three invariants from above. ∎

Theorem 4.3 (Trusted Election): Under the same assumptions as above, **apply** $dre\ L \hookrightarrow w \Diamond \mathcal{D}$ iff $w = \ulcorner \sum_{i=1\ldots n} I(v_i) \urcorner$ and $\mathcal{D} = \mathsf{evapp}\ \mathsf{evlam}\ \mathsf{ev!}\ \mathcal{D}_L$.

V. Logical Frameworks

The technique trace emitting computations is directly amenable to implementation using logical frameworks. We concentrate on the logical framework LF [HHP93] and the implementation in Twelf [PS99] reducing the need for trace checking to type checking. Our experiments regarding a real election are described in Section VI.

The logical framework LF serves as a meta-language for representing expressions, values, and traces. It is a dependently typed λ-calculus that offers the possibility to declare new constants on the type level, which are called type families if indexed by terms, and object level constants. LF subscribes to the judgments as types paradigm.

Any of these categories can be represented as type families `exp`, `val`, and `eval E V`, respectively. We have given the precise Twelf encodings of the values and expressions in Figure 4, and the encodings for traces in Figure 5. For simplicity the Twelf encodings use the same naming convention as Figure 3.

In Twelf, we write `type` for the kind of a type, and use curly braces `{x:exp}` for dependent types and dependent kinds. We write `->` (infix) if the dependency is vacuous. Furthermore, we write `[x:exp]` for λ-abstraction, `(x:exp)` for type ascription, and juxtaposition for application provided by the logical framework. We write $\vdash$ for the typing judgment in LF. Without loss of generality, we assume all LF derivations to be in normal form.

The representation of traces is *adequate*, in the sense that the representation function between syntactic categories and types in LF are bijections. This means, for example that if $\mathcal{D}$ is a valid trace for "e evaluates to v" then its representation is a closed normal form of type `eval E V`, and vice versa.

Theorem 5.1 (Adequacy):
1) Let e be an expression and let us write `e` for its encoding in LF. Then e with free variables among $x_1 \ldots x_n$ is valid if and only if `x1:val ... xn:val` $\vdash$ `e:exp`.
2) Let w be a value and let us write `w` for its encoding in LF. Then w with free variables among $x_1 \ldots x_n$ is valid if and only if `x1:val ... xn:val` $\vdash$ `w:val`.
3) Let $\mathcal{D}$ be a trace and let us write `D` for its encoding in LF. Also let e (`e`) and w (`w`) be an expression and value respectively. Then $e \hookrightarrow w \Diamond \mathcal{D}$ is valid if and only if $\vdash$ `D:eval e w`.

Proof: by mutual structural induction on LF canonical forms in one direction, and the structure of e, w, and $\mathcal{D}$ in the other. ∎

Furthermore, all bijections are compositional, in the sense that β-reduction of the logical framework mimics precisely substitution application of the language that we are are encoding.

```
a* : val.
b* : val.
z* : val.
s* : val -> val.
lam* : (val -> exp) -> val.
pair* : val -> val -> val.
cons* : val -> exp -> val.
nil* : val.

! : val -> exp.
a : exp.
b : exp.
case : exp -> exp -> exp -> exp.
z : exp.
s : exp -> exp.
lam : (val -> exp) -> exp.
app : exp -> exp -> exp.
pair : exp -> exp -> exp.
fst : exp -> exp.
snd : exp -> exp.
nil : exp.
cons : exp -> exp -> exp.
fold : exp -> exp
       -> (val -> val -> exp) -> exp.
```

Figure 4. Formalization: Values, Expressions

Theorem 5.2 (Compositionality):
1) Let e be an expression in which x occurs free and w a value. Furthermore, let us write `e` and `w` for the encoding of expression e and value w. The term that encodes $[w/x]e$ in Twelf is $\beta\eta$-equal to `([x:exp] e) w`.
2) Let $\mathcal{D} : e \hookrightarrow w$ be a trace in which $u : e' \hookrightarrow w'$ occurs free. Let us write `e`, `w`, `D`, `D'` for the respective encodings. The term that represents the instantiation of a $\mathcal{D}' : e' \hookrightarrow w'$ for u in $\mathcal{D}$: $[\mathcal{D}'/u]\mathcal{D}$, is $\beta\eta$ equal to `([u:eval e' w'] D) D'`.

Proof: by structural induction on the definition of substitution. ∎

These two facts might have the feel of be of academic interest only, but they are instrumental in creating trust into an electronic election based on trace emitting computations. They allow us to use the LF type checker to validate traces. The LF type checker is well-understood, it is small, and implementations run on different platforms and are implemented in different languages. There is one implementation in machine code that is verified by hand [AMSV02].

The advantage of using an LF type-checker to validate traces of electronic voting machines is due to the fact that it is independent of the concrete configuration of a voting machine, which means that its implementation does not change, even if the language of expressions, values, and traces is adapted from election to election. We will always

```
eval : exp -> val -> type.
ev!  : {V:val} eval (! V) V.
eva  : eval a a*.
evb  : eval b b*.
evcasea:
  {E:exp} {E1:exp} {W:val} {E2:exp}
    eval E a* -> eval E1 W
    -> eval (case E E1 E2) W.
evcaseb:
  {E:exp} {E2:exp} {W:val} {E1:exp}
    eval E b* -> eval E2 W
    -> eval (case E E1 E2) W.
evlam: {E:val -> exp}
    eval (lam ([x:val] E x))
         (lam* ([x:val] E x)).
evapp:
  {E1:exp} {E1':val -> exp} {E2:exp}
  {W2:val} {W:val}
    eval E1 (lam* ([x:val] E1' x))
    -> eval E2 W2 -> eval (E1' W2) W
    -> eval (app E1 E2) W.
evzero: eval z z*.
evsucc: {E:exp} {W:val}
    eval E W -> eval (s E) (s* W).
evpair:
  {E1:exp} {W1:val} {E2:exp} {W2:val}
    eval E1 W1 -> eval E2 W2
    -> eval (pair E1 E2) (pair* W1 W2).
evfst: {E:exp} {W1:val} {W2:val}
    eval E (pair* W1 W2)
    -> eval (fst E) W1.
evsnd: {E:exp} {W1:val} {W2:val}
    eval E (pair* W1 W2)
    -> eval (snd E) W2.
evnil: eval nil nil*.
evcons:
  {E1:exp} {W1:val} {E2:exp}
    eval E1 W1
    -> eval (cons E1 E2) (cons* W1 E2).
evfold1:
  {E1:exp} {E2:exp} {W:val}
  {E3:val -> val -> exp}
    eval E1 nil* -> eval E2 W
    -> eval (fold E1 E2 ([x] [y] E3 x y)) W.
evfold2:
  {E1:exp} {W11:val} {E12:exp} {E2:exp}
  {E3:val -> val -> exp} {W2:val} {W:val}
    eval E1 (cons* W11 E12)
    -> eval (fold E12 E2 ([x] [y] E3 x y)) W2
    -> eval (E3 W11 W2) W
    -> eval (fold E1 E2 ([x] [y] E3 x y)) W.
```

Figure 5. Formalization: Traces

use the same LF type checker, no matter if arithmetic is carried out in unary, binary, or even decimal notation.

But not only this. The trace of the election could theoretically be published after the election is completed. If the trace is constructed by randomly pasting the trace fragments casts by voters, it is not in violation of principle 1. It can then be checked by everyone, who understands how to implement an LF type checker.

In summary, logical frameworks enable us to minimize the trusted base of computation to a bare minimum, and relieve us of the fact that we need to trust the compiler, the semantics of the domain specific language, the hardware (to a certain extend), the graphical user interface, the configuration of a concrete machine, the fear of a virus process running in parallel, and gives us infinite confidence that the software that is running on an electronic voting machine is in fact the software that we expected to run. The LF signature specifies all there is to be known about the setup of the election.

VI. Experimental Results

We have built an experimental electronic voting machine based on the ideas of trace emitting computations. The machine was deployed for the election of a member to the board of the IT University of Copenhagen that took place October 9 and October 10, 2007. The election had a low participation record. The machine collected about 70 votes during the two days. Interestingly the majority of the votes were cast during lunch break, but before lunch, in an extremely short time span between 12:00 and 12:10. At times, 10 to 15 people were waiting in line. This means, that an effective processing of each vote was of the essence. Note, that for explanatory purposes, we describe an altered setup of the election. The setup for the real election was highly experimental.

The hardware of our electronic voting machine was provided by an Apple Macintosh Powerbook Pro. It was placed in an inaccessible corner of the voting booth, its keyboard sealed. A mouse was provided to interact with the graphical user interface. As we used USB sticks for authentication, we needed to give the user access to the USB port, which we did by a USB extension cord.

The process of voting was as follows. A prospective voter enters the voting place, authenticates using a student id card, and then obtains a USB stick which contains a software token that enables him or her to cast exactly one vote. We did not to solve the problem of making sure that the voter can only vote once. The USB stick is inserted into the USB extension cord, the machine awakens, the screen displays an array of radio buttons, including a *abstain to vote* choice, once a radio button is checked, the *cast the vote* button becomes enabled, and the voter can cast the vote.

Afterwards, the machine computed the new intermediate total, and recorded the steps it needed to do in form of a

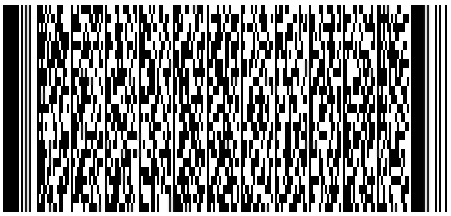

Figure 6. Ballot for candidate A

trace fragment. The fragment was then printed as physical evidence on a piece of paper in form of a barcode (see Figure 6). Once printing was complete, the machine reverted into waiting state to be activated by another prepared USB stick. The voter picked up the physical evidence, and on the way out, just as in a traditional election, dropped the ballot into the ballot box and discarded the now useless USB key into the box of used USB keys. The election staff continuously recycled the USB sticks with new tokens, so that they could be reused by the next voter.

During the time leading up to the election, we formalized the ballot in terms of who are the candidates, how many votes can be cast, etc. This formal description was then used to generate the Twelf signature and the user interface, which displayed for example, radio check boxes next to the candidate names. The text fields displayed along with the radio boxes is also taken from the formal specification. The generated Twelf signature is public and was studied, checked, and analyzed before the election.

The physical evidence produced by the voting machine, inspected by a third party scanning machine and cast by the voter was an LF object that reflected step by step what the voting machine has done. For elections with only finitely many valid ballots, one could imagine that the physical evidence contains abbreviations of the form $votea$ or $voteb$, but this should be seen as a special case for a very simple election.

Figure 6 depicts a Pdf417 barcode that contains the full expanded information trace fragment, necessary to count a ballot

```
[L:val] [X:val] [Y:val]
  [r:eval (fold (! L) (pair z z)
    ([x5:val] [x6:val]
       case (! x5) (pair (s (fst (! x6)))
                         (snd (! x6)))
    (pair (fst (! x6)) (s (snd (! x6))))))
  (pair* X Y)]
evfold2 ev! r (evcasea ev! (evpair (evsucc
  (evfst ev!)) (evsnd ev!))).
```

The leading λ-abstractions of this trace fragment abstract away all personal identifying information from a ballot. It enables us to treat ballots of the same kind uniformly. That we can use this trick here is in fact a deep consequence of the Compositionality Theorem 5.2.

The election was a two day election, which confronted us with the question of what to do with the electronic voting machine over night. Because of the technology employed here, we could afford the machine running over night, without human observation. Following the ideas outlined in the paper, we took the sealed ballot box containing the partial computation trace and stored it over night at a safe location because we were sure that when reconstructing the trace, we would notice if the election was tempered with.

At the end of the second day. We terminated the electronic voting software presenting us with the result Twelf target type (here shortened to only 6 votes):

```
eval (fold (! (cons* b* (! (cons* a*
  (! (cons* b* (! (cons* b* (! (cons* a*
  (! (cons* a* (! nil*))))))))))))
  (pair z z)
  ([x:val] [x5:val]
    case (! x) (pair (s (fst (! x5)))
                     (snd (! x5)))
      (pair (fst (! x5))
            (s (snd (! x5))))))
  (pair* (s* (s* (s* z*)))
         (s* (s* (s* z*))))
```

After scanning all of the ballots, we pasted them together into one large LF object, reconstructed its type using Twelf, and checked the two types for equivalence modulo $\beta\eta$-conversion. The trace emitting computation technique certified the outcome of the election.

If one uses a unary definition of natural numbers as in this example, it is as difficult to read out the result of an election from the type returned as it would have been to hand-count the votes in the first place. This problem can be easily fixed in Twelf by using a binary, or decimal representation of numbers in the definition of expressions. However, such a choice would have complicated this presentation, the inspection and analysis of the Twelf signature.

VII. Analysis

During this election Iben Lewinsky and Mikkel Sesøe Sørensen have conducted a rigorous empirical study [LS08]. Using a mixed methods approach, three different groups have been surveyed and the results show that the technical implementation is of less importance to the immediate trust relations than factors, such as system usability and media attention. Also, the study reveals a wide readiness for the adoption of e-voting systems in Denmark, as well as a high apparent level of trust in the capabilities and security of such systems; the former attributable to an inherent high level of trust in Danish authorities. We have therefore concluded that the empirical studies have a significant bias that is difficult to quantify. To alleviate this problem we plan to supplement

the empirical studies by theoretical work using modern trust models.

VIII. Scalability

An election is a process that consists of many parts, where casting a vote is just one among many. Although this paper focuses on this particular part, we speculate that trace emitting computations scale to the other parts as well. Consider for examples the summing up results of individual voting machines.

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be the evidence for the evaluation of $\mathbf{apply}\ dre\ L_1 \hookrightarrow w_1 \Diamond \mathcal{D}_1$ and $\mathbf{apply}\ dre\ L_2 \hookrightarrow w_2 \Diamond \mathcal{D}_2$. In order to compute the sum of w_1 and w_2, we need to ensure that this computation is being executed on a machine. Based on the definition of addition, which we omit here, the new trace $\mathcal{D}$ can be constructed from traces $\mathcal{D}_1$ and $\mathcal{D}_2$. As above, this construction can be validated ahead of time, and is purely mechanical.

Another example is the computation of which party gets how many seats in parliament. And again, after extending the language displayed in Figure 3 by a case construct for natural numbers, we can encode the algorithm that converts tallies into seats. And again we need to use two machines. One which does the computation, and the other that constructs the trace for it. Eventually, one obtains both, and under the independence assumption, problems with the computation cannot remain unnoticed.

IX. Conclusion

We have proposed a technique of trace emitting computations, with which illustrate how to generate trust in electronic voting software. The diamond of trust illustrates just how trust is generated: The electronic voting machine executes computation and provides a trace whose individual parts are either inspected by the voter (to make sure that it corresponds to the voters intent) or automatically generated (by an open, verified, and formalized procedure). Before the result of an election is final, a small trusted piece of software checks that the trace is in fact a valid trace of the computation of the conjectured result. Only if this test is positive, the election is validated.

In future work, we will develop modern trust models, study the nature and implication of the assumption that the voting and trace machines are assumed to be "independent". In particular, we will study the prevention of covet information (as part of the physical evidence) to flow between the two machines.

Acknowledgments: I would like to thank the two students who implemented a prototype of the electronic voting machine, Erik Cederstrand and Kenneth Sjøholm and the two other students who helped me conduct a real election with this technology, Iben Lewinsky and Mikkel Sesøe Sørensen. I also want to thank the anonymous reviewers for their valuable suggestions.

References

[AMSV02] A. Appel, N. Michael, A. Stump, and R. Virga. A trustworthy proof checker. Technical Report TR-647-02, Department of Computer Science, Princeton University, 2002.

[CHF07] Joseph A. Calandrino, J. Alex Halderman, and Edward W. Felten. Machine-assisted election auditing. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, 2007. USENIX Association.

[Eve07] Sarah P. Everett. *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection*. PhD thesis, Rice University, 2007.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[LS08] Iben Lewinsky and Mikkel Selsøe Sørensen. Trust in e-voting systems. Master's thesis, IT University of Copenhagen, Department of Computer Science, Copenhagen University, 2008.

[Nec97] George C. Necula. Proof-carrying code. In Neil D. Jones, editor, *Conference Record of the 24th Symposium on Principles of Programming Languages (POPL'97)*, pages 106–119, Paris, France, January 1997. ACM Press.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[RS07] Ronald L. Rivest and Warren D. Smith. Three voting protocols: Threeballot, vav, and twin. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop*, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[SBR01] David Jefferson Shuki Bruck and Ronald Rivest. A modular voting architecture ("frogs"). Technical Report VTP Working Paper #3, Caltech/MIT Voting Technology Project, August 2001.