Type-Based Verification of Electronic Voting Protocols

Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei and Cyrille Wiedling

Formal Methods Seminar

22nd May 2015

INRIA, Rennes, France
Introduction: E-voting evolution

Canada: Since 2004 at the Provincial level. (EVM and (later) Internet voting.)

USA: EVM used for legally binding vote since 1996.

Brazil: legally binding e-vote with EVM since 2000.

India: legally binding e-voting with EVM since 2002.

Estonia: 2005, first legally binding vote using Internet.

But also: Norway, France, Poland, ...

Planned in: Mexico, China, Spain, ...

Estonia: 2005, first legally binding vote using Internet.
Introduction: E-voting, in theory

Electronic Voting provides:

• **Convenience**
  Better accessibility, remote voting…

• **Efficiency**
  Computers are tallying faster than humans.

• **Reliability**
  Computers are more accurate than humans.

• **Trust**
  Everything is ensured by cryptography.
Introduction: E-voting, in theory

- Coercion-Resistance
- Privacy
- Verifiability
- Eligibility

E-voting promises better security
But…

Things can go wrong.

- Diebold Machines in the U.S.
  (Candice Hoke, 2008)
- Paperless EVM in India.
  (A. Halderman, R. Gonggrijp, 2010)

So, we need proofs! automated proofs!
Introduction: Tools can’t make it!

Automated proofs often take place in the **symbolic approach**.

<table>
<thead>
<tr>
<th><strong>Computational</strong></th>
<th><strong>Symbolic</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>More realistic model</td>
<td>Weaker guarantees</td>
</tr>
<tr>
<td><strong>Strong guarantees</strong></td>
<td>Abstract model</td>
</tr>
<tr>
<td>Attacker modeled by probabilistic polynomial-time Turing Machine</td>
<td>cryptographic primitives as function symbols</td>
</tr>
<tr>
<td><strong>Tedious proofs</strong></td>
<td>Easier proofs often automated</td>
</tr>
<tr>
<td>cryptographic primitives as polynomial algorithms</td>
<td></td>
</tr>
</tbody>
</table>

There are **numerous tools** that can already perform automated proofs.

- aKiSs
- Tamarin
- SPEC
- APTE
- AVISPA
- Scyther
- ProVerif

E-Voting protocols often include **too many** different cryptographic primitives!
We needed something different!

Something that could handle equivalence-based properties.

Then, the rF* type-checker [Barthe et al. POPL'14] appears.
(with the ability to verify equivalence-based properties)

We'll see that in details a bit later…

So we asked the question:

Can type-checkers be used to verify (automatically) e-voting protocols?
But first, what are type-systems?

- A type is a description that characterizes the expected form of the result of a computation.

If $e$ is an expression, and we consider the following typing:

$$e : \text{int}$$

This is a typing judgement asserting that the value of $e$ is of type int.

- NB: It will also checks consistency.

$$2 + 1 : \text{int} \quad \text{true : int} \quad 2 + \text{true} : \text{int}$$
• A type-system is a set of types and constructors used to describe the expected behavior of a program.

• The goal of the type-checker is to verify the different typing judgements and see whether they are true or not.

  This is done by using rules from which it can derive the assertion.

• Basically, enforcing that \( e : \tau \) means that:

  - \( e \) is well-typed, i.e. correctly derived of type \( \tau \) using the rules.
  
  - When \( e \) is evaluated, its value is described by \( \tau \).
• What kind of rules?

\[ n : \text{int} \]

**Function mapping \( \tau_1 \) to \( \tau_2 \).**

\[ e_1 : \tau_1 \rightarrow \tau_2 \]
\[ e_2 : \tau_1 \]
\[ e_1 e_2 : \tau_2 \]

• How does the type-checker to verify: \( 2 + 1 : \text{int} \)?

\[ + : \text{int} \rightarrow \text{int} \rightarrow \text{int} \]
\[ 2 : \text{int} \]
\[ + 2 : \text{int} \rightarrow \text{int} \]
\[ 1 : \text{int} \]
\[ + 2 1 : \text{int} \]
Of course, we need a type-system (a bit) more elaborated to be able to express electronic-voting protocols.

But this in not an issue…

How does it work ?

Soundness result : If a program type-checks, then it is safe. (In a presence of an arbitrary attacker.)
One interesting point:
SMT Solvers do not have any problem with AC-properties.

So... Can type-checkers be used to verify (automatically) e-voting protocols?

We decided to give it a go:

• Developing a logical theory to guide type-checker in proving interesting security properties like privacy and verifiability.

• Analyzing an existing e-voting protocol as an applied example.
An Outline of what follows

I. Helios, our running example.

II. Verifiability
   1. Individual Verifiability
   2. Universal Verifiability
   3. End-to-end Verifiability

III. Privacy

What’s next, Doc?
Helios: Running example

- Web-based electronic voting system
  Try it at [https://vote.heliosvoting.org/](https://vote.heliosvoting.org/)!

- Two existing versions: homomorphic encryption VS mixnets.

- Already used for several elections.
  (Louvain-la-Neuve University, IACR* Board, …)

*International Association for Cryptologic Research
Helios (Simplified)

Alice  
Bob  
Charlie

Bulletin Board

\[
\{v_A, r_A\}_{pk(E)}
\]

\[
\{v_B, r_B\}_{pk(E)}
\]

\[
\{v_C, r_C\}_{pk(E)}
\]

• \(pk(E)\): public key. The private one is shared among trustees. (All should collaborate to perform decryption of the tally.)

• The tally is computed using homomorphic encryption (El-Gamal). (The encrypted result is \(\{v_A + v_B + v_C, r_A + r_B + r_C\}_{pk(E)}\).)

• Only the final result is encrypted, implying vote privacy.
Helios (Simplified)

A bit overly simplified…

\[
\begin{align*}
\{v_A, r_A\}_{\text{pk}(E)} + \text{zkp}(v_A = 0 \text{ or } 1) \\
0/1
\end{align*}
\]

\[
\begin{align*}
\{v_B, r_B\}_{\text{pk}(E)} + \text{zkp}(v_B = 0 \text{ or } 1) \\
0/1
\end{align*}
\]

\[
\begin{align*}
\{v_C, r_C\}_{\text{pk}(E)} + \text{zkp}(v_C = 0 \text{ or } 1) \\
\text{[X]} \quad \text{[X]} \quad \text{[X]}
\end{align*}
\]

- A zero-knowledge proof is attached to the ciphertext.
  (It may also provide a proof to the correctness of the final tally.)

- Using ZKP, Helios satisfies end-to-end verifiability.
Verifiability: Let’s have an intuition of it!

There are three different notions of verifiability:

• **Individual verifiability**: Each voter can check that their ballot is on the bulletin board.

• **Universal verifiability**: Any observer can verify that the announced result corresponds to the ballots published on the bulletin board.

• **End-to-end verifiability**: The result matches with the votes intended by the voters.
Individual Verifiability

How to prove individual verifiability using a type-system?

\[ \text{Voter}(id, v) = \begin{array}{ll}
\text{assume Vote}(id, v); & \text{send}(net, b) \\
\text{let } r = \text{new}() \text{ in} & \text{let } bb = \text{recv}(net) \text{ in} \\
\text{let } b = \text{enc}(pk, v, r) \text{ in} & \text{if } b \in bb \text{ then} \\
\text{assume MyBallot}(id, v, b); & \text{assert VHappy}(id, v, bb)
\end{array} \]

• We introduce three predicates: \textbf{Vote, MyBallot} and \textbf{VHappy}.

• We define when the predicate VHappy should be verified:

\[ \text{assume VHappy}(id, v, b) \iff \text{Vote}(id, v) \land \exists b \in bb \text{ MyBallot}(id, v, b) \]

• We can prove that if such an annotated protocol type-checks…

\textbf{Then it guarantees individual verifiability}!

We used type-checker F* [Swamy and al. ICFP’11]
Universal Verifiability

How is made the tally?

- A step of **sanitization** where we remove duplicates and invalid ballots from the bulletin board. \( bb \mapsto vbb \)  

  (Don’t remove the honest votes !)

- A step of **counting** where all the votes contained in ballots listed in \( vbb \) are counted.

We need some predicates…

\[
\text{GoodCount}(vbb, r) \quad \text{GoodSan}(bb, vbb)
\]

assume \( \text{JudgeHappy}(bb, r) \iff \exists vbb \ (\text{GoodSan}(bb, vbb) \land \text{GoodCount}(vbb, r)) \)
We now use these predicates to encode a Judge…

\[
\text{Judge}(bb, r) = \begin{align*}
&\text{let } vbb = \text{recv}(net) \text{ in} \\
&\text{let } zkp = \text{recv}(net) \text{ in} \\
&\text{if } vbb = \text{remDuplicates}(bb) \land \text{check}_{zkp}(zkp, vbb, r) \text{ then} \\
&\text{assert } \text{JudgeHappy}(bb, r)
\end{align*}
\]

• We can prove that if such an annotated protocol type-checks…

Then it guarantees universal verifiability!

We used type-checker F* [Swamy and al. ICFP'11]
End-To-End Verifiability

We repeat the same scheme we used for individual or universal verifiability.

New predicate:

\[
\text{assume EndToEnd} \iff \exists \ bb, r, id_1, \ldots, id_n, v_1, \ldots, v_n. \\
(JudgeHappy(bb, r) \land VHappy(id_1, v_1, bb) \land \cdots \land VHappy(id_n, v_n, bb)) \\
\implies \exists \ rlist \ . \ r = \rho(rlist) \land \{|v_1, \ldots, v_n|\} \subseteq_m rlist
\]

However this is difficult to enforce using a type-system.

Nevertheless, does this definition ring any bell?

Idea: individual + universal = end-to-end

But...
Clash-Attacks [Küsters et al. S&P’12]

- Alice and Bob will vote the same way.
- Machines of Alice and Bob are corrupted by Charlie.
- One vote can be discarded and replace by another one…

without Alice nor Bob noticing it!
NoClash Property

Yes, another predicate!

\[
\text{assume NoClash } \iff \forall id_1, id_2, v_1, v_2, b . \\
\text{MyBallot}(id_1, v_1, b) \land \text{MyBallot}(id_2, v_2, b) \\
\implies id_1 = id_2 \land v_1 = v_2
\]

Two distinct honest voters will never consider the same ballot to contain their vote.

• We can prove that if such an annotated protocol type-checks…

  Then it guarantees that there are no clashes!

  We used type-checker F* [Swamy and al. ICFP’11]

• Then, we have an interesting result:

  \text{Individual Verif. + Universal Verif. + NoClash = End-to-End Verif.}
Verifiability: Conclusion

• We defined a way to prove individual and universal verifiability using type-systems (F*).

• We applied this methodology to Helios and verified that it holds.

• Using the NoClash predicate, we have a way to prove end-to-end verifiability using type-systems.

• Thanks to previous results, it also holds for Helios.
Privacy: Definition

What is privacy in an electronic-voting protocol?

Idea 1: Should my vote remain secret?

Well… We need to reveal votes in order to get the result…

Idea 2: Should no one see the difference if I change my vote?

∀\(\mathcal{P}\) indistinguishable from \((\mathcal{P}(\mathcal{P}))(\approx)\)

In the case of unanimity, the difference is kinda… obvious.
Idea 3: Should no one see the difference if two honest voters swap their votes?

**Definition** (S. Delaune, S. Kremer, M. Ryan, 2009)

\[ \forall \mathcal{D} (P((a,b)) \equiv (P((a,b)))) \]

Observational Equivalence
• rF* can be used to enforce observational equivalence.

• To do so, it implements relational refinements which allows to reason about two protocol runs:

\[ x : T\{|F|\} \]

• To specify that a value is the same in both runs, we use eq-types:

\[ \text{eq } T \triangleq x : T\{Lx = Rx\} \]

Value of x in the first execution.

• All inputs/outputs should be typed with eq types.
Privacy: Typing it! (with rF*)

\[ x : \text{bytes}\{ | Lx = v_1 \land Rx = v_2 | \} \]

Alice \( v_A = \)

\[
\text{let } b_A = \text{create\_Ballot}_A(v_A) \text{ in } \\
\text{send}(c_A, b_A)
\]

Bob \( v_B = \)

\[
\text{let } b_B = \text{create\_Ballot}_B(v_B) \text{ in } \\
\text{send}(c_B, b_B)
\]

• We add a corrupted voter, who also submits a ballot:
  \( b_C = \{ v_C, r_C \}_{pk(E)} \)

• The corrupted voter submits the same thing at each execution, thus:
  \( v_C \text{ is of type } x : \text{eq bytes} \)

• Finally, the result, after decryption, is:
  \[ v_A + v_B + v_C \]

Result is an eq bytes, we can publish it!

\[ v_1 + v_2 + v_C \approx v_2 + v_1 + v_C \]
Finally…

• New definitions for individual, universal, end-to-end verifiability and privacy that are enforceable by mechanized type-based analysis.

• A theorem proving that end-to-end verifiability is enforced by both individual and universal verifiability and no-clash property.

• Using F* and rF*, we proved the security properties of Helios.

• Can we apply it to other protocols?
That's all folks!

Questions?