Type-Based Verification of Electronic Voting Protocols

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

Formal Methods Seminar

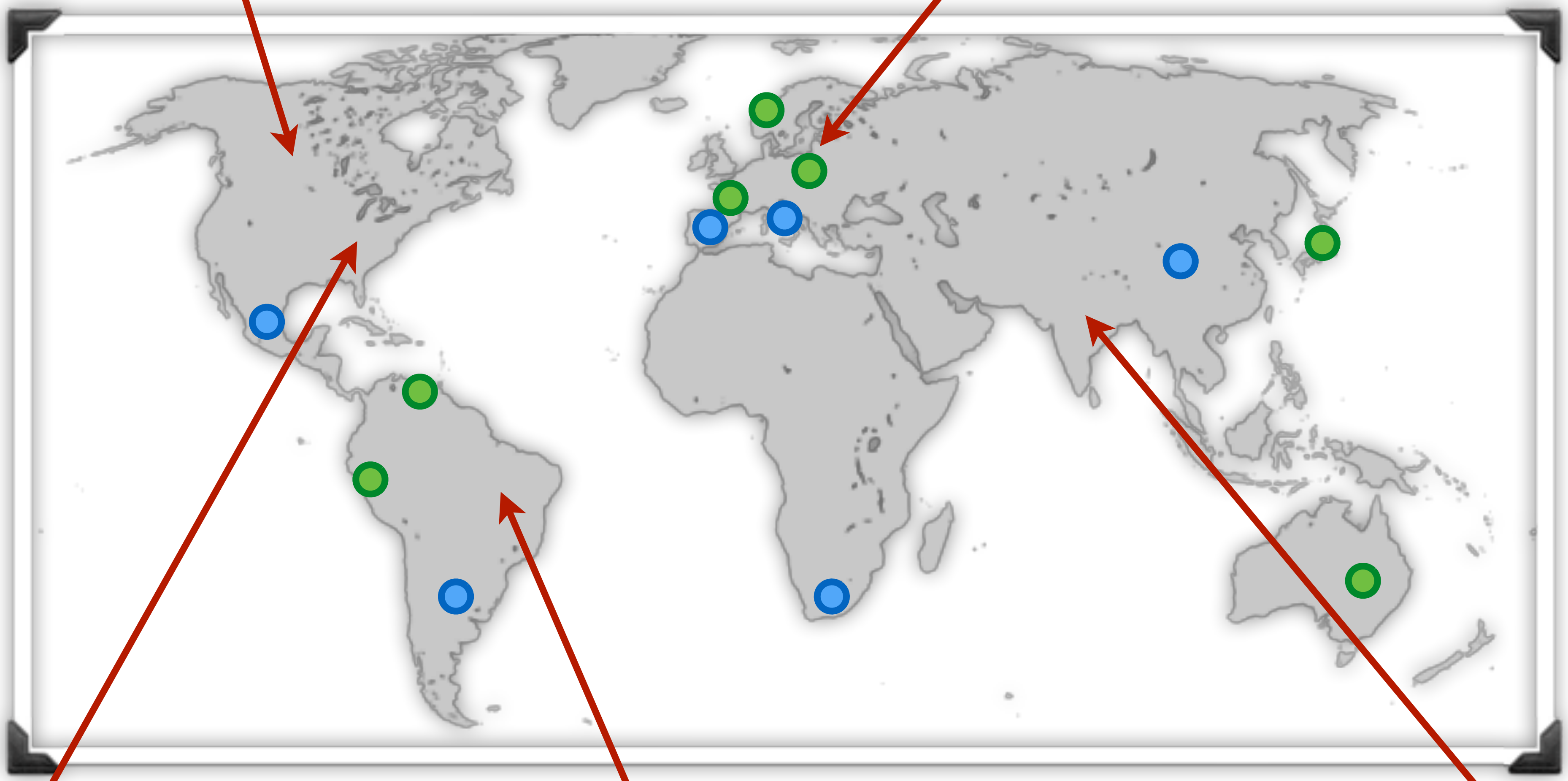
22nd May 2015
INRIA, Rennes, France

CISPA
Center for IT-Security, Privacy
and Accountability

Introduction: E-voting evolution

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

Estonia : 2005, first legally binding vote using Internet.



But also :
Norway
France,
Poland,
...

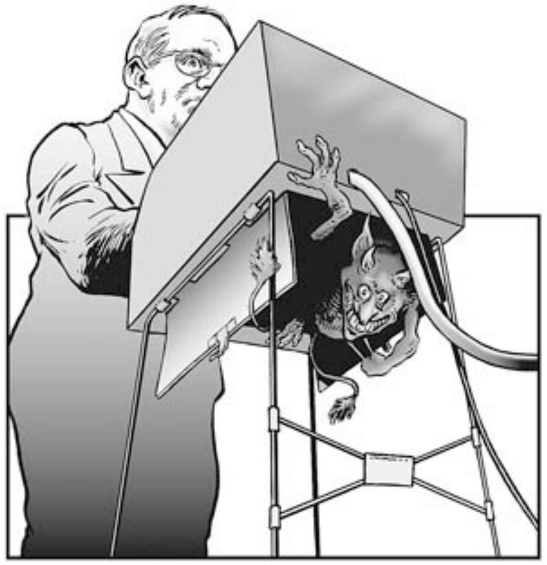
Planned in :
Mexico,
China,
Spain,
...

USA : EVM used for legally binding vote since 1996.

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

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

Introduction: E-voting, in theory



Electronic Voting Machines

Internet Voting

Electronic Voting provides :

- **Conveniency**

  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

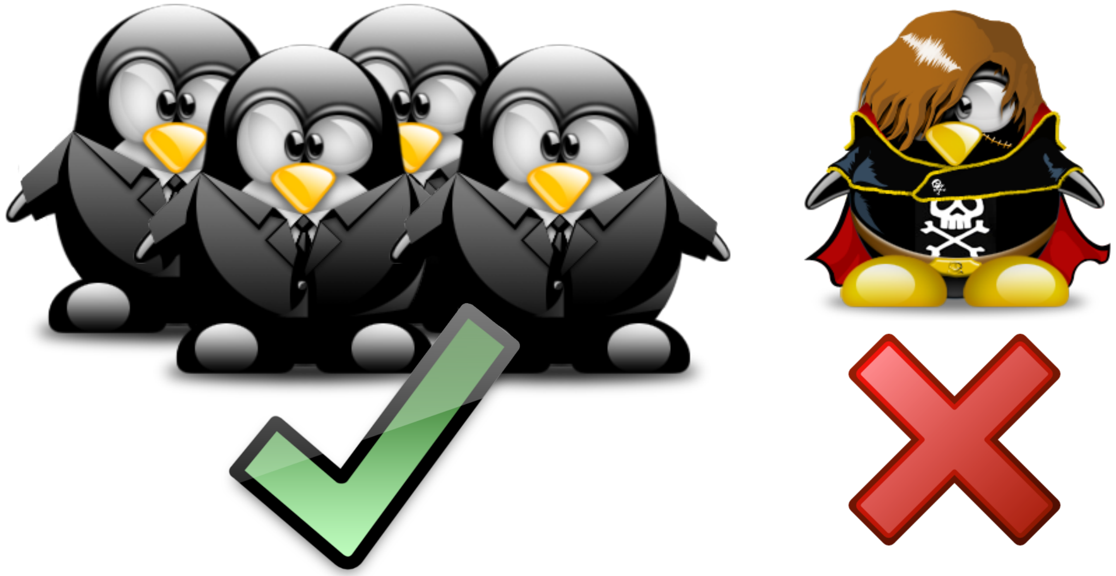
Coercion-Resistance

Verifiability

E-voting promises
better security

Privacy

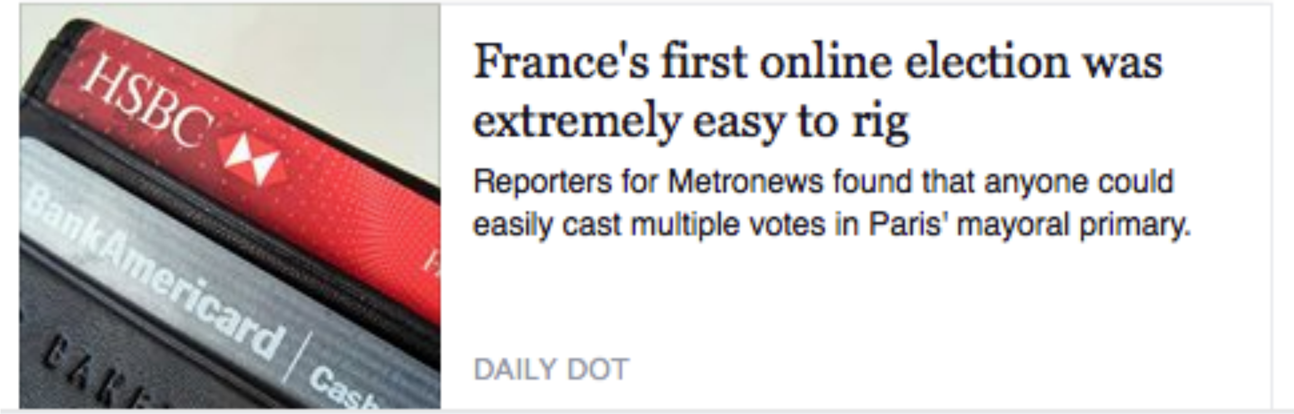
Eligibility



Introduction: E-voting, in practice

But…

Things can go wrong.



France's first online election was extremely easy to rig

Reporters for Metronews found that anyone could easily cast multiple votes in Paris' mayoral primary.

DAILY DOT

- 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**.

Computational

More realistic model

Strong guarantees

Attacker modeled by probabilistic polynomial-time Turing Machine

Tedious proofs

cryptographic primitives as polynomial algorithms

Symbolic

Weaker guarantees

Abstract model

Attacker modeled by deduction rules

cryptographic primitives as function symbols

Easier proofs often automated

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

aKiSs

Tamarin

APTE

SPEC

AVISPA

Scyther

ProVerif

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

Introduction: A new tool ?

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

?

Introduction: Type-Systems

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.

$2 + 1 : \text{int}$ ✓ $\text{true} : \text{int}$ ✗ $2 + \text{true} : \text{int}$ ✗

Introduction: Type-Systems

- 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 wether 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 τ using the rules.

  - When e is evaluated, its value is described by τ.

Introduction: Type-Systems

- What kind of rules ?

Function mapping τ_1 to τ_2 .

$$\frac{}{n : \text{int}} \qquad\qquad \frac{e_1 : \tau_1 \to \tau_2 \qquad e_2 : \tau_1}{e_1 e_2 : \tau_2}$$

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

$$\dfrac{\dfrac{\dfrac{}{+ : \text{int} \to \text{int} \to \text{int}} \qquad \dfrac{}{2 : \text{int}}}{+\ 2 : \text{int} \to \text{int}} \qquad \dfrac{}{1 : \text{int}}}{+\ 2\ 1 : \text{int}}$$

✓

Introduction: Type-Systems

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 ?

Type-Checker

SMT Solver

Protocol Specification

Proof Obligations

Result

Soundness result : If a program type-checks, then it is safe.
(In a presence of an arbitrary attacker.)

Introduction: Type-Systems

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 ?**

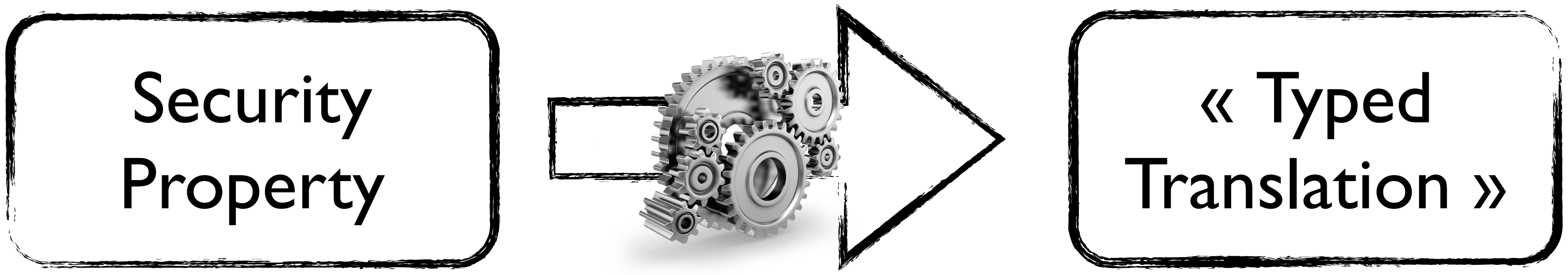
We decided to give it a go:

- 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.



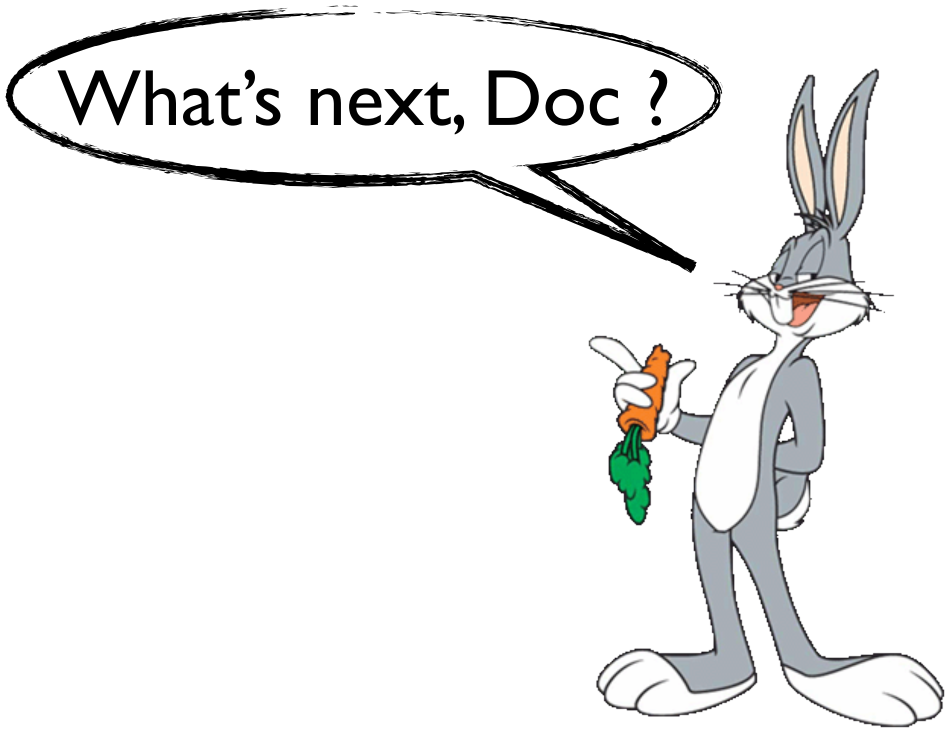
Security Property

« Typed Translation »

II. Verifiability

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

III. Privacy



Helios: Running example

Trust the vote.

- Web-based electronic voting system

  Try it at 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

$\{v_A, r_A\}_{\mathsf{pk(E)}}$

Bob

$\{v_B, r_B\}_{\mathsf{pk(E)}}$

Bulletin Board

Charlie

$\{v_C, r_C\}_{\mathsf{pk(E)}}$

- $\mathsf{pk(E)}$: 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 $\{v_A + v_B + v_C, r_A + r_B + r_C\}_{\mathsf{pk(E)}}$.)
- Only the final result is encrypted, implying **vote privacy**.

Helios (Simplified)

A bit overly simplified…

Alice

$\{v_A, r_A\}_{\mathsf{pk(E)}} + \mathsf{zkp}(v_A = 0 \text{ or } 1)$

$0/1$

Bob

Bulletin Board

$\{v_B, r_B\}_{\mathsf{pk(E)}} + \mathsf{zkp}(v_B = 0 \text{ or } 1)$

$0/1$

Charlie

$\{v_C, r_C\}_{\mathsf{pk(E)}} + \mathsf{zkp}(v_C = 0 \text{ or } 1)$

~~100!~~

- 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 is 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 ?

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

- We introduce three predicates : **Vote**, MyBallot and 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…

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…

$\mathsf{GoodCount}(vbb, r)$

$\mathsf{GoodSan}(bb, vbb)$

$$\text{assume } \mathsf{JudgeHappy}(bb, r) \iff \exists\, vbb\ (\mathsf{GoodSan}(bb, vbb) \land \mathsf{GoodCount}(vbb, r))$$

Universal Verifiability

We now use these predicates to encode a Judge…

$$\begin{aligned}\text{Judge}(bb, r) = \quad & \text{let } vbb = \text{recv}(net) \text{ in} \\ & \text{let } zkp = \text{recv}(net) \text{ in} \\ & \text{if } vbb = \text{remDuplicates}(bb) \wedge \text{check\_zkp}(zkp, vbb, r) \text{ then} \\ & \text{assert JudgeHappy}(bb, r)\end{aligned}$$

- 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 :

$$\text{assume EndToEnd} \iff \exists\; bb, r, id_1, \ldots, id_n, v_1, \ldots, v_n .$$

$$(\text{JudgeHappy}(bb, r) \wedge \text{VHappy}(id_1, v_1, bb) \wedge \cdots \wedge \text{VHappy}(id_n, v_n, bb))$$

$$\implies \exists\; rlist\ .\ r = \rho(rlist) \wedge \{|v_1, \ldots, v_n|\} \subseteq_m rlist$$

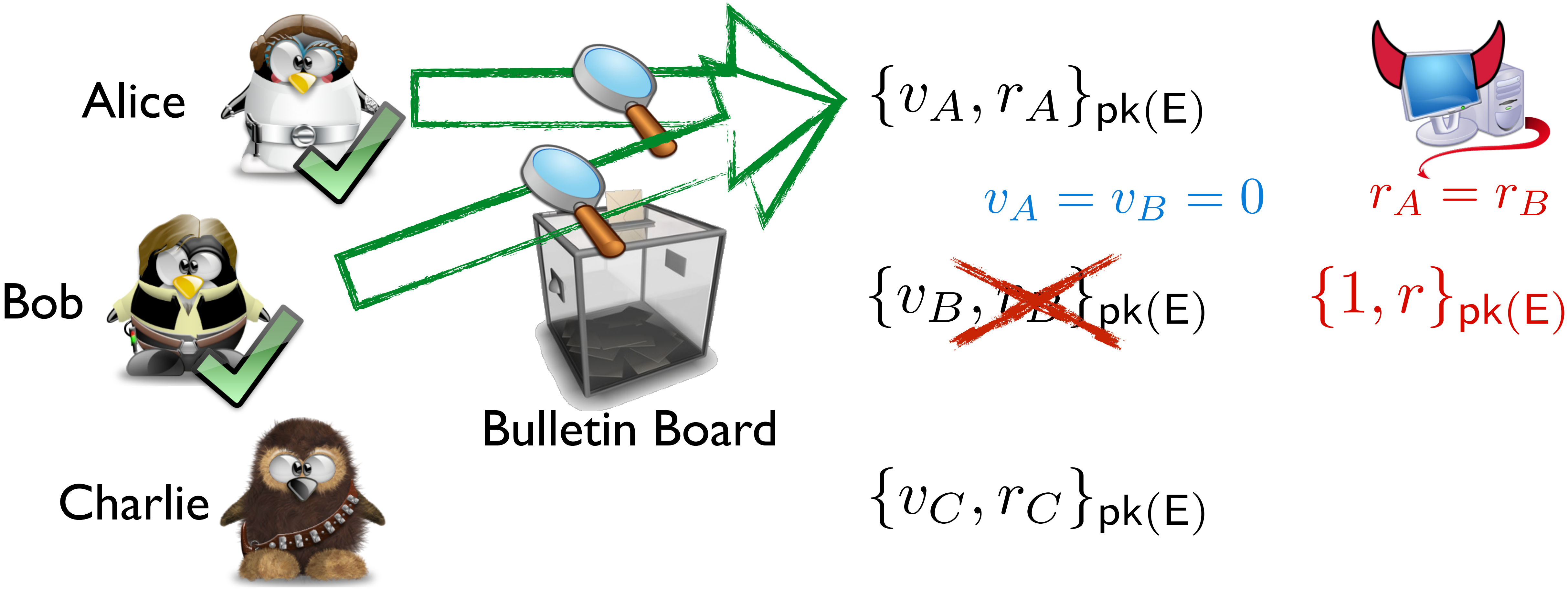
However this is **difficult to enforce** using a type-system.

Nevertheless, does this definition ring any bell ?

Idea: individual + universal = end-to-end

But…

Clash-Attacks [Küsters et al. S&P'12]



Alice

Bob

Charlie

Bulletin Board

$\{v_A, r_A\}_{\mathsf{pk(E)}}$

$v_A = v_B = 0$ $r_A = r_B$

$\{v_B, \cancel{r_B}\}_{\mathsf{pk(E)}}$ $\{1, r\}_{\mathsf{pk(E)}}$

$\{v_C, r_C\}_{\mathsf{pk(E)}}$

- 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 !

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

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 clashs !

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

- Then, we have an interesting result :

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 ? REJECTED

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 ? REJECTED

In the case of unanimity, the difference is kinda… obvious.

Privacy: Definition

Idea 3:

Should no one see the difference if **two honest voters** swap their votes ? ✓

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

Observational Equivalence

Privacy: Using rF* to prove it

- 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*)

$x : \text{bytes}\{|Lx = v_1 \wedge Rx = v_2|\}$

$x : \text{bytes}\{|Lx = v_2 \wedge Rx = v_1|\}$

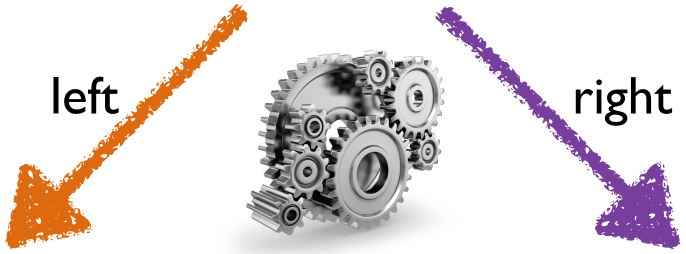
Alice $v_A =$
let $b_A = \text{create\_Ballot}_A(v_A)$ in
send(c_A, b_A)

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

- We add a corrupted voter, who also submits a ballot : $b_C = \{v_C, r_C\}_{\text{pk(E)}}$
- The corrupted voter submits the same thing at each execution, thus :

v_C is of type $x : \text{eq bytes}$

- Finally, the result, after decryption, is : $v_A + v_B + v_C$



$$v_1 + v_2 + v_C \approx v_2 + v_1 + v_C$$

Result is an eq bytes, we can publish it !

Conclusion

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 ?