Formal verification of electronic voting systems

Véronique Cortier, CNRS, Loria (Nancy, France)

June 8th, 2023

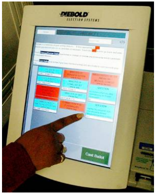
Why e-voting?

- Convenient
  ⟶ for voters: vote from home, or abroad
  ⟶ for authorities: easier to record and tally votes
- More “democracy”
  ⟶ complex tally process (Condorcet, STV, IRV)
  ⟶ can be used more often
  ⟶ complex legal rules
  (a voter may vote from any place in their state)
- Many protocols have been proposed:
  Helios, Belenios, Civitas, Prêt-à-Voter, Selene, CHVote, sElect, StarVote, . . .

Two main families for electronic voting

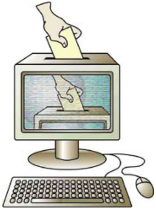
Voting machines

- Voters attend a polling station;
- Standard authentication (id cards, etc.)



Internet Voting

- Voters vote from home;
- Using their own computer (or phone, tablet, ...)



Internet voting is used in various countries

- France: National parliament for the French expats (2012, 2022)
- Australia: New South Wales state (2021, more than 650 000 votes cast by Internet)
- Estonia: local elections (since 2005), national parliamentary elections (2007, 2011, 2015, 2019)
- Switzerland: several trials, a demanding and evolving regulation since 2013
- Canada: local election in Ontario (since 2003) and Nova Scotia (since 2006)

...banned in other countries !

- Netherland: 2008, electronic voting is abolished (voting machine and Internet)
- Germany: 2009, the voting machines (Nedap) are rejected, do not comply with the constitution

  It must be possible for a citizen to check the main steps of a voting process, with no special expertise.

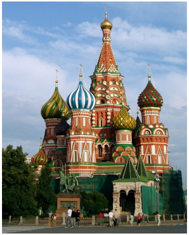
- Norway: trials ended in 2013

  The fear of voters that their vote might become public may undermine the democratic process.

Widely used in non-political election

- professional elections
- associations
- administration councils
- scientific councils

Numerous attacks !

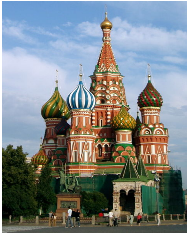


Elections in Moscow [P. Gaudry]

- ballots posted on a blockchain (why?)
- bug bounty program

3 keys of 256 bits $\neq$ 1 key of 768 bits

Numerous attacks !



Elections in Moscow [P. Gaudry]

- ballots posted on a blockchain (why?)
- bug bounty program

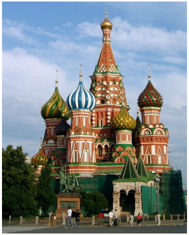
3 keys of 256 bits $\neq$ 1 key of 768 bits

Swiss context

- open specification, open source code
- call for public scrutiny
- multiple elections in one round

SWISS POST

Numerous attacks !



Elections in Moscow [P. Gaudry]

- ballots posted on a blockchain (why?)
- bug bounty program

3 keys of 256 bits $\neq$ 1 key of 768 bits

Swiss context

- open specification, open source code
- call for public scrutiny
- multiple elections in one round

Privacy breach with A. Debant and P. Gaudry

- possibility to (silently) add an extra ballot box, with just Alice' ballot
- a generous bug bounty 😀

What is a good voting system?

Confidentiality of the votes

Vote privacy

"No one should know how I voted"

Confidentiality of the votes

Vote privacy

"No one should know how I voted"

Better: Receipt-free / Coercion-resistant

"No one should know how I voted,
even if I am willing to tell my vote! "

Confidentiality of the votes

Vote privacy

"No one should know how I voted"

Better: Receipt-free / Coercion-resistant

"No one should know how I voted,
even if I am willing to tell my vote! "

- vote buying
- coercion

ebay

Silk Road
anonymous marketplace

Confidentiality of the votes

Vote privacy

"No one should know how I voted"

Better: Receipt-free / Coercion-resistant

"No one should know how I voted,
even if I am willing to tell my vote! "

- vote buying
- coercion

ebay

Silk Road
anonymous marketplace

Everlasting privacy: no one should know my vote, even when the cryptographic keys will be eventually broken.

Verifiability

Individual Verifiability: a voter can check that

- cast as intended: their ballot contains their intended vote
- recorded as cast: their ballot is in the ballot box.

Universal Verifiability: everyone can check that

- tallied as recorded: the result corresponds to the ballot box.
- eligibility: ballots have been casted by legitimate voters.

You should verify the election,
not the system.

Verifiability

Individual Verifiability: a voter can check that

- cast as intended: their ballot contains their intended vote
- recorded as cast: their ballot is in the ballot box.

Universal Verifiability: everyone can check that

- tallied as recorded: the result corresponds to the ballot box.
- eligibility: ballots have been casted by legitimate voters.

You should verify the election,
not the system.

Even better: accountability

- the system tells whom to blame
- eases dispute resolution

And many more properties

- Availability: servers available at any time
- Accessibility: easy to use, adapted to people with various issues
- ...

I should not be able to prove how I voted, yet I should be able to check that my vote has been counted...

I should not be able to prove how I voted, yet I should be able to check that my vote has been counted...

Let's see how this can be realized.

Voting protocol Belenios

- variant of Helios, designed by Ben Adida
- developed at Loria, teams Pesto and Caramba (P. Gaudry)
  Developer: Stéphane Glondu
- used in 2000+ elections, with a total of 100 000+ voters

`http://www.belenios.org/`

- confidentiality of the votes
- verifiability of the voting process
  → The ballot box is public at any time.
  → All the operations (tally, ...) can be checked by anyone.

Building blocks: cryptography

Threshold decryption

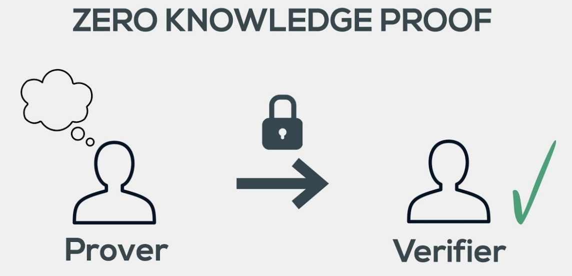
- Each trustee computes their secret key
- The n trustees jointly compute the public key pk
- Decryption with t out of the n keys:
  t out of n trustees suffice to produce decryption shares, that yield the plaintext

Threshold decryption

- Each trustee computes their secret key
- The n trustees jointly compute the public key pk
- Decryption with t out of the n keys:
  t out of n trustees suffice to produce decryption shares, that yield the plaintext

→ The decryption key is never present on a single computer, neither during the key generation nor the decryption!

Zero-Knowledge proofs



Examples

- Possibility to prove that an encrypted message is either *a* or *b*

$$\{m\}_k \quad \mathit{Proof}(m = a \text{ or } m = b)$$

- Possibility to prove that the decryption is correct

$$c, m \quad \mathit{Proof}(\mathrm{dec}_k(c) = m)$$

How Belenios works (simplified)

Phase 1: vote

pk(E)

Ballot Box

Alice	$\{v_A\}_{\mathsf{pk}(E)}$	$v_A = 0$ or 1
Bob	$\{v_B\}_{\mathsf{pk}(E)}$	$v_B = 0$ or 1
Chris	$\{v_C\}_{\mathsf{pk}(E)}$	$v_C = 0$ or 1

pk(E): public key, the private keys are shared among the authorities.

How Belenios works (simplified)

Phase 1: vote

$\xrightarrow{id, \{v\}^{r}_{\mathrm{pkE}}}$

pk(E)

Ballot Box

Alice	$\{v_A\}_{\mathrm{pk}(E)}$
Bob	$\{v_B\}_{\mathrm{pk}(E)}$
Chris	$\{v_C\}_{\mathrm{pk}(E)}$

$v_A = 0$ or 1

$v_B = 0$ or 1

$v_C = 0$ or 1

pk(E): public key, the private keys are shared among the authorities.

How Belenios works (simplified)

Phase 1: vote

pk(E)

Ballot Box

Alice	$\{v_A\}_{\mathrm{pk}(E)}$	$v_A = 0$ or 1
Bob	$\{v_B\}_{\mathrm{pk}(E)}$	$v_B = 0$ or 1
Chris	$\{v_C\}_{\mathrm{pk}(E)}$	$v_C = 0$ or 1
David	$\{v_D\}_{\mathrm{pk}(E)}$	$v_D = 0$ or 1

pk(E): public key, the private keys are shared among the authorities.

How Belenios works (simplified)

Phase 1: vote

pk(E)

Ballot Box

Alice	$\{v_A\}_{\mathsf{pk}(E)}$	$v_A = 0$ or 1
Bob	$\{v_B\}_{\mathsf{pk}(E)}$	$v_B = 0$ or 1
Chris	$\{v_C\}_{\mathsf{pk}(E)}$	$v_C = 0$ or 1
David	$\{v_D\}_{\mathsf{pk}(E)}$	$v_D = 0$ or 1
...	...	

Phase 2: Tally - homomorphic encryption (El Gamal)

$$\{v_1\}_{\mathsf{pk}(E)} \times \cdots \times \{v_n\}_{\mathsf{pk}(E)} = \{v_1 + \cdots + v_n\}_{\mathsf{pk}(E)} \quad \text{since } g^a \times g^b = g^{a+b}$$

→ Only the final result needs to be decrypted! And proved.

pk(E): public key, the private keys are shared among the authorities.

Oversimplified!

$$\xrightarrow{id, \{v\}^r_{\mathrm{pkE}}}$$

pk(E)

Ballot Box

Alice	$\{v_A\}_{\mathrm{pk}(E)}$	$v_A = 0$ or 1
Bob	$\{v_B\}_{\mathrm{pk}(E)}$	$v_B = 0$ or 1
Chris	$\{v_C\}_{\mathrm{pk}(E)}$	$v_C = 0$ or 1
David	$\{v_D\}_{\mathrm{pk}(E)}$	
...	...	

Result: $\{v_A + v_B + v_C + v_D + \cdots\}_{\mathrm{pk}(E)}$

Oversimplified!

$$\xrightarrow{id, \{v\}^{r}_{\mathrm{pkE}}}$$

pk(E)

Ballot Box		
Alice	$\{v_A\}_{\mathrm{pk}(E)}$	$v_A = 0$ or 1
Bob	$\{v_B\}_{\mathrm{pk}(E)}$	$v_B = 0$ or 1
Chris	$\{v_C\}_{\mathrm{pk}(E)}$	$v_C = 0$ or 1
David	$\{v_D\}_{\mathrm{pk}(E)}$	$v_D = 100$
...	...	

Result: $\{v_A + v_B + v_C + 100 + \cdots\}_{\mathrm{pk}(E)}$

A voter could cheat!

Oversimplified!

$$\xrightarrow{id, \{v\}^r_{\mathrm{pkE}}}$$

$\mathrm{pk}(E)$

Ballot Box

Alice	$\{v_A\}_{\mathrm{pk}(E)}$	$v_A = 0$ or 1
Bob	$\{v_B\}_{\mathrm{pk}(E)}$	$v_B = 0$ or 1
Chris	$\{v_C\}_{\mathrm{pk}(E)}$	$v_C = 0$ or 1
David	$\{v_D\}_{\mathrm{pk}(E)}$	~~$v_D = 100$~~
...	...	

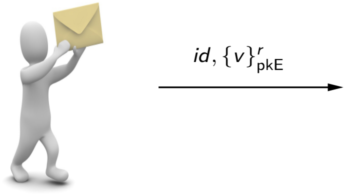
Result: $\{v_A + v_B + v_C + v_D + \cdots\}_{\mathrm{pk}(E)}$

~~A voter could cheat!~~

Use a zero-knowledge proof

$$\{v_D\}_{\mathrm{pk}(E)}, \mathrm{Proof}\{v_D = 0 \text{ or } v_D = 1\}$$

Still oversimplified



$id, \{v\}^{r}_{\mathrm{pkE}}$

$\mathrm{pk}(E)$

Ballot box

Alice	$\{v_A\}_{\mathrm{pk}(E)}$
Bob	$\{v_B\}_{\mathrm{pk}(E)}$
Chris	$\{v_C\}_{\mathrm{pk}(E)}$
...	...
...	

Still oversimplified

$\xrightarrow{id, \{v\}^{r}_{\mathrm{pkE}}}$

$\mathrm{pk}(E)$

Ballot box	
Alice	$\{v_A\}_{\mathrm{pk}(E)}$
Bob	$\{v_B\}_{\mathrm{pk}(E)}$
Chris	$\{v_C\}_{\mathrm{pk}(E)}$
...	$\{1\}_{\mathrm{pk}(E)}$
...	$\{1\}_{\mathrm{pk}(E)}$

The ballot box could add ballots!

Still oversimplified

$id, \{v\}^{r}_{\mathrm{pkE}}$ →

$\mathrm{pk}(E)$ $\mathrm{vk}(cred_3), \mathrm{vk}(cred_1), \mathrm{vk}(cred_2), \ldots$

Ballot box

Alice	$\{v_A\}_{\mathrm{pk}(E)}$
Bob	$\{v_B\}_{\mathrm{pk}(E)}$
Chris	$\{v_C\}_{\mathrm{pk}(E)}$
...	
...	

~~The ballot box could add ballots!~~

1. During the setup phase, a Registrar generates private signing keys, one for each voter

Still oversimplified

$\xrightarrow{id, \{v\}^{r}_{\mathrm{pkE}}}$

$\mathrm{pk}(E)$ $\mathrm{vk}(cred_3), \mathrm{vk}(cred_1), \mathrm{vk}(cred_2), \ldots$

Ballot box

Alice	$[\{v_A\}_{\mathrm{pk}(E)}]_{\mathrm{sk}(cred_1)}$
Bob	$[\{v_B\}_{\mathrm{pk}(E)}]_{\mathrm{sk}(cred_2)}$
Chris	$[\{v_C\}_{\mathrm{pk}(E)}]_{\mathrm{sk}(cred_3)}$
...	
...	

~~The ballot box could add ballots!~~

1. During the setup phase, a Registrar generates private signing keys, one for each voter
2. The voters sign their ballot with a "credential" they have received (a credential = a right to vote)

Some additional features

Many cryptographic features:

- blank votes: select 3 to 5 candidates among 10 OR vote blank
- threshold decryption: 5 out 7 trustees are sufficient to decrypt
- support both homomorphic encryption and mixnets
  - rank candidates: Condorcet, STV
  - score candidates: Majority Judgement

Multi-languages: English, French, German, Spanish, Czech, Norwegian, Portuguese, Greek, Italian, ...

→ Just add yours! (easy, Weblate platform)

How Belenios is used?

since 2020:

- about 1500 elections / year on our voting platform
- 100 000+ ballots cast in total
- about 25 independant voting servers
- initial users
  - universities for councils representatives, hiring commitees
  - many sport associations (chess, handball) or other associations
  - companies for representatives
- but also:
  - FDP party (Germany)
  - European Court of Accounts (ECA)
  - Université Libre de Bruxelles (ULB)
  - Italian Scouts Federation

Distribution of Belenios

There are two ways for running an election with Belenios.

1. Install your own server. Belenios is an open-source software, available at:

   `https://gitlab.inria.fr/belenios/belenios`

2. Use our online voting platform:

   `https://belenios.loria.fr/admin`

   - the administrator can set up an election and manage the election authorities
   - the decryption trustees can generate (locally) their private key in their browser (also in the threshold mode)
   - the registrar can generate (locally) all the credentials in their browser. They then need to send the credentials to the voters (typically by email).

Formal analysis of e-voting systems

Why a formal analysis of an e-voting system?

Formal analysis of e-voting systems

Why a formal analysis of an e-voting system?

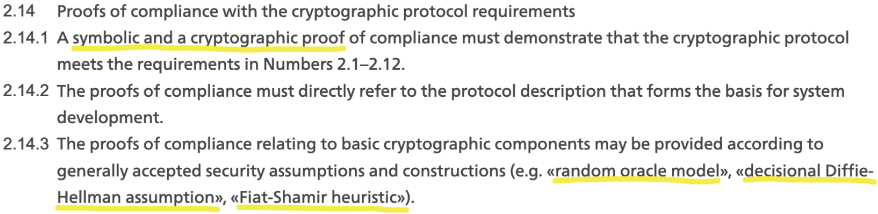
⟶ Because formal methods can find attacks before implementations
⟶ Now a current practice for many protocols (TLS, 5G, ...)

Formal analysis of e-voting systems

Why a formal analysis of an e-voting system?

⟶ Because formal methods can find attacks before implementations
⟶ Now a current practice for many protocols (TLS, 5G, ...)

→ Legal requirements in Switzerland to provide **symbolic and cryptographic proofs** of e-voting protocols.



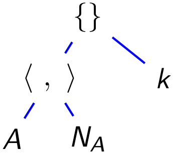
2.14 Proofs of compliance with the cryptographic protocol requirements

2.14.1 A symbolic and a cryptographic proof of compliance must demonstrate that the cryptographic protocol meets the requirements in Numbers 2.1–2.12.

2.14.2 The proofs of compliance must directly refer to the protocol description that forms the basis for system development.

2.14.3 The proofs of compliance relating to basic cryptographic components may be provided according to generally accepted security assumptions and constructions (e.g. «random oracle model», «decisional Diffie-Hellman assumption», «Fiat-Shamir heuristic»).

Two main models for security

	Formal approach	Computational approach
Messages	 terms	0101000101110101 1101010110101010 0011101011101101 bitstrings
Encryption		algorithm
Adversary	idealized	any polynomial algorithm
Guarantees	some attacks missed	stronger
Proof	often automatic	mostly by hand difficult for complex protocols

Messages

Messages are abstracted by terms.

Agents : $a, b, \ldots$

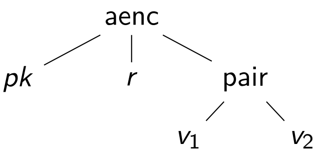
Keys : $k_1, k_2, \ldots$

Ciphertext : $\mathrm{aenc}(pk, r, m)$

Nonces : $n_1, n_2, \ldots$

Concatenation : $\mathrm{pair}(m_1, m_2)$
denoted simply (m_1, m_2) in ProVerif

Example: The encrypted message $\mathrm{aenc}(pk, r, \mathrm{pair}(v_1, v_2))$ is represented by:



Intuition: only the structure of the message is kept.

Model for cryptographic primitives

Projection

$$\pi_1(\mathsf{pair}(x, y)) = x$$

$$\pi_2(\mathsf{pair}(x, y)) = y$$

Asymmetric and symmetric encryption

$$\mathsf{adec}(\mathsf{aenc}(\mathsf{pk}(y), z, x), y) = x$$

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x$$

Model for cryptographic primitives

Projection

$$\pi_1(\mathsf{pair}(x,y)) = x$$

$$\pi_2(\mathsf{pair}(x,y)) = y$$

Asymmetric and symmetric encryption

$$\mathsf{adec}(\mathsf{aenc}(\mathsf{pk}(y),z,x),y) = x$$

$$\mathsf{dec}(\mathsf{enc}(x,y),y) = x$$

Zero knowledge proof: proof of valid vote

$$\mathsf{aenc}(\mathsf{pk},r,m), \mathsf{ZKP}(m = 0 \text{ OR } m = 1)$$

$$\begin{aligned} \mathsf{Valid}(\mathsf{ZKP}(\mathsf{aenc}(\mathsf{pk},r,0),\mathsf{pk},r),\mathsf{aenc}(\mathsf{pk},r,0),\mathsf{pk}) &= \mathsf{ok} \\ \mathsf{Valid}(\mathsf{ZKP}(\mathsf{aenc}(\mathsf{pk},r,1),\mathsf{pk},r),\mathsf{aenc}(\mathsf{pk},r,1),\mathsf{pk}) &= \mathsf{ok} \end{aligned}$$

Syntax for processes

The grammar of processes is as follows:

$$
\begin{array}{l}
P, Q, R := \\
\quad 0 \\
\quad \texttt{if } M_1 = M_2 \texttt{ then } P \texttt{ else } Q \\
\quad \text{let } x = M \text{ in } P \\
\quad \text{in}(c, x); P \\
\quad \text{out}(c, N); P \\
\quad \text{new } n; P \\
\quad P \mid Q \\
\quad !P
\end{array}
$$

Syntax of ProVerif, a dialect of the applied-pi calculus
[AbadiFournet01]

Example: Belenios light

$$\begin{array}{lcl} A & \to S & id_A, \mathrm{aenc}(pkE, r_A, v_0) \\ B & \to S & id_B, \mathrm{aenc}(pkE, r_B, v_1) \\ S & \to & \{v_0, v_1\} \end{array}$$

r_a random number generated by A.
r_b random number generated by B.

Example: Belenios light

$$\begin{array}{lll}
A & \rightarrow S & id_A, \mathrm{aenc}(pkE, r_A, v_0) \\
B & \rightarrow S & id_B, \mathrm{aenc}(pkE, r_B, v_1) \\
S & \rightarrow & \{v_0, v_1\}
\end{array}$$

r_a random number generated by A.
r_b random number generated by B.

We need to model two processes:

- one corresponding to the role of a voter
- one corresponding to the role of the server

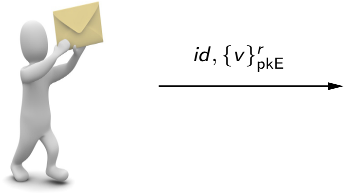
Role of a voter

$$\xrightarrow{id, \{v\}^{r}_{\mathsf{pkE}}}$$

free c : channel.

let $Voter(\mathsf{pkE}, \mathsf{Vote}, id, \mathsf{cauth}) =$

Role of a voter

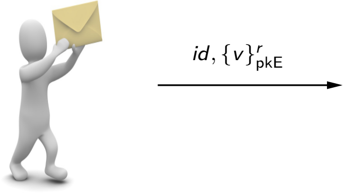


$id, \{v\}^{r}_{\mathsf{pkE}}$

```
free c : channel.

let Voter(pkE, Vote, id, cauth) =
   new r : bitstring;
   let b = (id, aenc(pkE, r, Vote)) in
```

Role of a voter



$id, \{v\}^{r}_{\mathrm{pkE}}$

```
free c : channel.

let Voter(pkE, Vote, id, cauth) =
   new r : bitstring;
   let b = (id, aenc(pkE, r, Vote)) in
   out(cauth, b);
   out(c, b).
```

Security properties

Secrecy query

$$\textsc{Not}\ \mathsf{attacker}(s)$$

Security properties

Secrecy query

$$\textsc{Not}\ \ \mathsf{attacker}(s)$$

Correspondence query $F_1, \ldots, F_n \Rightarrow \phi$

Example:

$$\mathsf{Voted}(id, v, r) \wedge \mathsf{EndTally} \Rightarrow \mathsf{Counted}(v)$$

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

Idea 1: An attacker should not learn the value of my vote.

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

Idea 1: An attacker should not learn the value of my vote.

But everyone knows 0 and 1!

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

~~Idea 1: An attacker should not learn the value of my vote.~~

Idea 2: An attacker cannot see the difference when voters are different $\text{Voter}(A, 0) \approx \text{Voter}(B, 0)$

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

~~Idea 1: An attacker should not learn the value of my vote.~~

Idea 2: An attacker cannot see the difference when voters are different $\text{Voter}(A, 0) \approx \text{Voter}(B, 0)$

Who voted might be public

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

~~Idea 1: An attacker should not learn the value of my vote.~~

~~Idea 2: An attacker cannot see the difference when voters are different~~ ~~$\mathrm{Voter}(A, 0) \approx \mathrm{Voter}(B, 0)$~~

Idea 3: An attacker cannot see the difference when I vote 0 or 1.

$$\mathrm{Voter}(A, 0) \approx \mathrm{Voter}(A, 1)$$

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

~~Idea 1: An attacker should not learn the value of my vote.~~

~~Idea 2: An attacker cannot see the difference when voters are different~~ ~~$\mathsf{Voter}(A, 0) \approx \mathsf{Voter}(B, 0)$~~

Idea 3: An attacker cannot see the difference when I vote 0 or 1.

$$\mathsf{Voter}(A, 0) \approx \mathsf{Voter}(A, 1)$$

- The attacker always sees the difference since the tally differs.
- Unanimity does break privacy.

How to model vote privacy in symbolic models?

How to state formally:

"No one should know my vote (0 or 1)"?

~~Idea 1: An attacker should not learn the value of my vote.~~

~~Idea 2: An attacker cannot see the difference when voters are different~~ ~~$\mathsf{Voter}(A, 0) \approx \mathsf{Voter}(B, 0)$~~

~~Idea 3: An attacker cannot see the difference when I vote 0 or 1.~~

$$\cancel{\mathsf{Voter}(A, 0) \approx \mathsf{Voter}(A, 1)}$$

Idea 4: An attacker cannot see when votes are swapped.

$$\mathsf{Voter}(A, 0) \mid \mathsf{Voter}(B, 1) \approx \mathsf{Voter}(A, 1) \mid \mathsf{Voter}(B, 0)$$

S. Kremer & M. Ryan

ProVerif: automatic analysis of protocols

Developed by Bruno Blanchet and Vincent Cheval

Performs very well in practice!

- Works on most of existing protocols in the literature
- Is also used on industrial protocols (e.g. TLS, Signal, ...)
- used to pass Swiss requirements on voting
  - Neuchâtel/Scytl protocol [C., Turuani 2018]
  - CHVote protocol [C., Turuani 2019]

ProVerif: automatic analysis of protocols

Developed by Bruno Blanchet and Vincent Cheval

Performs very well in practice!

- Works on most of existing protocols in the literature
- Is also used on industrial protocols (e.g. TLS, Signal, ...)
- used to pass Swiss requirements on voting
  - Neuchâtel/Scytl protocol [C., Turuani 2018]
  - CHVote protocol [C., Turuani 2019]

→ ProVerif translates processes in applied pi-calculus into Horn clauses (first-order logic).

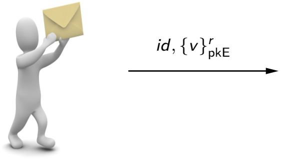
Intruder

Horn clauses perfectly reflects the attacker symbolic manipulations on terms.

$$
\begin{array}{llcll}
\forall x \forall y & I(x), I(y) & \Rightarrow & I(\mathrm{enc}(x,y)) & \text{encryption} \\
\forall x \forall y & I(\mathrm{enc}(x,y)), I(y) & \Rightarrow & I(x) & \text{decryption} \\
\forall x \forall y & I(x), I(y) & \Rightarrow & I(< x, y >) & \text{concatenation} \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(x) & \text{first projection} \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(y) & \text{second projection}
\end{array}
$$

Protocol as Horn clauses

```
let Voter(pkE, Vote, id, cauth) =
   new r : bitstring;
   let b = (id, aenc(pkE, r, Vote)) in
   event Voted(id, Vote, r)
   out(cauth, b);
   out(c, b).
```



Each action of the protocol is translated into logical implications.

$$\begin{array}{lcl} \forall v \quad I(v) & \Rightarrow & I(\langle id, \mathrm{aenc}(\mathrm{pkE}, r(v), v\rangle)) \\ \forall v \quad I(v) & \Rightarrow & \mathrm{Voted}(id, v, r(v)) \end{array}$$

Security reduces to consistency

secure?

$\rightsquigarrow$

$$
\begin{array}{lrcl}
\forall x \forall y & I(x), I(y) & \Rightarrow & I(< x, y >) \\
\forall x \forall y & I(x), I(y) & \Rightarrow & I(\mathrm{enc}(x, y)) \\
\forall x \forall y & I(\mathrm{enc}(x, y)), I(y) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(y)
\end{array}
$$

$$
\begin{array}{llcl}
\forall v & I(v) & \Rightarrow & I(\langle id, \mathrm{aenc}(\mathrm{pkE}, r(v), v \rangle)) \\
\forall v & I(v) & \Rightarrow & \mathrm{Voted}(id, v, r(v))
\end{array}
$$

Security reduces to consistency

secure?

$\rightsquigarrow$

$$\begin{array}{llcl}
 & & & \textsc{Not}\ I(\text{secret}) \\
\forall x \forall y & I(x), I(y) & \Rightarrow & I(< x, y >) \\
\forall x \forall y & I(x), I(y) & \Rightarrow & I(\text{enc}(x, y)) \\
\forall x \forall y & I(\text{enc}(x, y)), I(y) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(y)
\end{array}$$

$$\begin{array}{llcl}
\forall v & I(v) & \Rightarrow & I(\langle id, \text{aenc}(\text{pkE}, r(v), v\rangle) \\
\forall v & I(v) & \Rightarrow & \text{Voted}(id, v, r(v))
\end{array}$$

Does not yield a contradiction ?

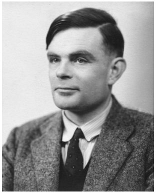
(i.e. consistent theory ?)

How to know if a set of formula is consistent ?

Hilbert's program (1928)
"Entscheidung Problem"

David Hilbert

It is undecidable! (1936)
→ There is no algorithm that answers this question.



Alan Turing

(at a time with no computers)

Security reduces to consistency: but undecidable!

secure?

$\boxed{\text{All this for nothing?}}$

$$
\begin{array}{lrcl}
\forall x \forall y & I(x), I(y) & \Rightarrow & I(< x, y >) \\
\forall x \forall y & I(x), I(y) & \Rightarrow & I(\mathrm{enc}(x, y)) \\
\forall x \forall y & I(\mathrm{enc}(x, y)), I(y) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(y)
\end{array}
$$

$$
\begin{array}{llcl}
\forall v & I(v) & \Rightarrow & I(\langle id, \mathrm{aenc}(\mathrm{pkE}, r(v), v\rangle) \\
\forall v & I(v) & \Rightarrow & \mathrm{Voted}(id, v, r(v))
\end{array}
$$

Security reduces to consistency: but undecidable!

secure?

All this for nothing?

$$\rightsquigarrow$$

$$\textsc{Not}\ I(\text{secret})$$

$$\begin{array}{lrcl}
\forall x \forall y & I(x), I(y) & \Rightarrow & I(< x, y >) \\
\forall x \forall y & I(x), I(y) & \Rightarrow & I(\text{enc}(x, y)) \\
\forall x \forall y & I(\text{enc}(x, y)), I(y) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(x) \\
\forall x \forall y & I(< x, y >) & \Rightarrow & I(y)
\end{array}$$

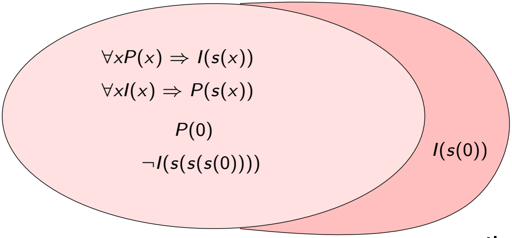
$$\begin{array}{llcl}
\forall v & I(v) & \Rightarrow & I(\langle id, \text{aenc}(\text{pkE}, r(v), v\rangle) \\
\forall v & I(v) & \Rightarrow & \text{Voted}(id, v, r(v))
\end{array}$$

Does not yield a contradiction ?

(i.e. consistent theory ?)

A standard technique: resolution

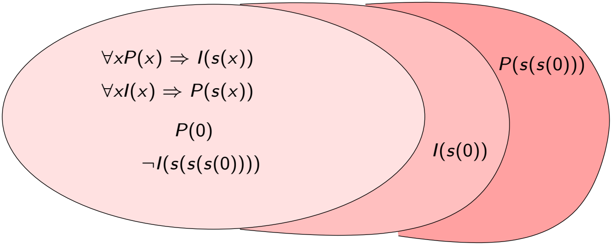
Idea: add logical consequences . . .


$\forall x P(x) \Rightarrow I(s(x))$
$\forall x I(x) \Rightarrow P(s(x))$
$P(0)$
$\neg I(s(s(s(0))))$
$I(s(0))$

... until a contradiction is found.

A standard technique: resolution

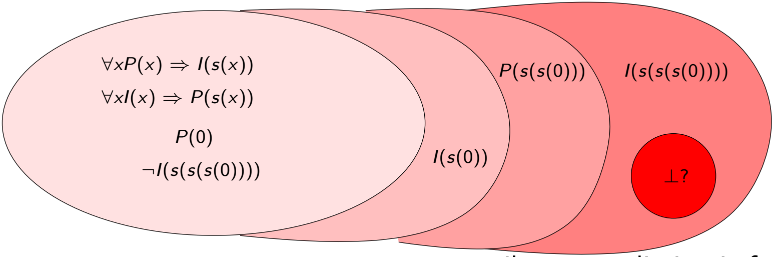
Idea: add logical consequences . . .


$\forall x P(x) \Rightarrow I(s(x))$
$\forall x I(x) \Rightarrow P(s(x))$
$P(0)$
$\neg I(s(s(s(0))))$
$I(s(0))$
$P(s(s(0)))$

... until a contradiction is found.

A standard technique: resolution

Idea: add logical consequences . . .

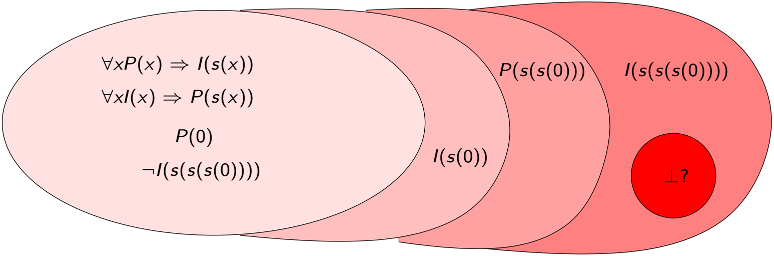


$\forall x P(x) \Rightarrow I(s(x))$
$\forall x I(x) \Rightarrow P(s(x))$
$P(0)$
$\neg I(s(s(s(0))))$
$I(s(0))$
$P(s(s(0)))$
$I(s(s(s(0))))$
$\perp$?

... until a contradiction is found.

A standard technique: resolution

Idea: add logical consequences . . .


$\forall x P(x) \Rightarrow I(s(x))$
$\forall x I(x) \Rightarrow P(s(x))$
$P(0)$
$\neg I(s(s(s(0))))$
$I(s(0))$
$P(s(s(0)))$
$I(s(s(s(0))))$
$\bot$?

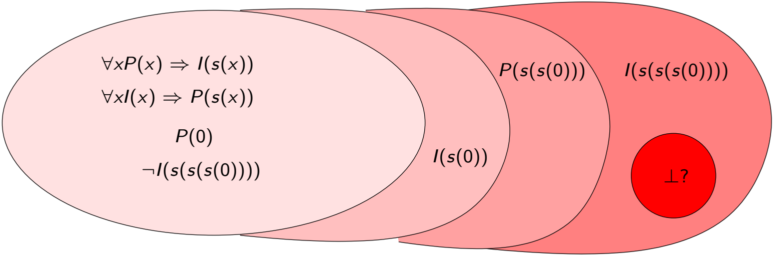
... until a contradiction is found.

Ideally, we need a method (a strategy) which is:

- correct: adds formula that are indeed consequences
- complete: finds a contradiction (if it exists)
- in a finite number of steps

A standard technique: resolution

Idea: add logical consequences . . .



$\forall x P(x) \Rightarrow I(s(x))$
$\forall x I(x) \Rightarrow P(s(x))$
$P(0)$
$\neg I(s(s(s(0))))$
$I(s(0))$
$P(s(s(0)))$
$I(s(s(s(0))))$
$\perp$?

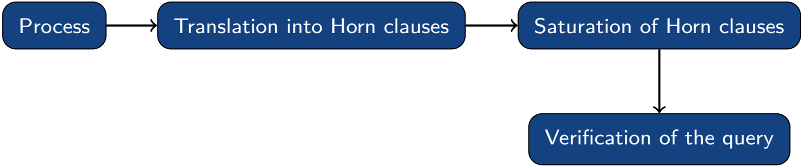
... until a contradiction is found.

Ideally, we need a method (a strategy) which is:

- correct: adds formula that are indeed consequences
- complete: finds a contradiction (if it exists)
- ~~in a finite number of steps~~ undecidable fragment

ProVerif

- Implements a correct procedure (that may not terminate or just stop without answer).
- Based on a resolution strategy well adapted to protocols.


Process
Translation into Horn clauses
Saturation of Horn clauses
Verification of the query

Binary resolution

$$\frac{H \Rightarrow C \quad F, H' \Rightarrow C'}{H\sigma, H'\sigma \Rightarrow C'\sigma} \text{ with } \sigma \text{ substitution s.t. } C\sigma = F\sigma$$

- correct
- but adds too many clauses (never terminates)

Binary resolution

$$\frac{H \Rightarrow C \quad F, H' \Rightarrow C'}{H\sigma, H'\sigma \Rightarrow C'\sigma} \text{ with } \sigma \text{ substitution s.t. } \begin{array}{r} C\sigma = F\sigma \\ F \neq I(x) \end{array}$$

- correct
- but adds too many clauses (never terminates)

ProVerif's strategy:

- do not resolve on $I(x)$
  Theorem: it remains refutationally complete
- well crafted order of resolution

Example

$$\mathcal{C} = \{\neg I(s), \quad I(k_1), \quad I(\{s\}_{\langle k_1,k_1\rangle}), \quad I(\{x\}_y), I(y) \Rightarrow I(x), \qquad I(x), I(y) \Rightarrow I(\langle x,y\rangle)$$

$$\dfrac{\neg I(s) \qquad \dfrac{\dfrac{I(\{s\}_{\langle k_1,k_1\rangle}) \quad I(\{x\}_y), I(y) \Rightarrow I(x)}{I(\langle k_1,k_1\rangle) \Rightarrow s} \qquad \dfrac{I(k_1) \qquad \dfrac{I(k_1) \quad I(x), I(y) \Rightarrow I(\langle x,y\rangle)}{I(y) \Rightarrow I(\langle k_1,y\rangle)}}{I(\langle k_1,k_1\rangle)}}{I(s)}}{\bot}$$

But it is not terminating!

$$\cfrac{\cfrac{I(y) \Rightarrow I(\langle s, y\rangle) \qquad \cfrac{I(y) \Rightarrow I(\langle s, y\rangle) \qquad \cfrac{I(s) \qquad \cfrac{I(s) \quad I(x), I(y) \Rightarrow I(\langle x, y\rangle)}{I(y) \Rightarrow I(\langle s, y\rangle)}}{I(\langle s, s\rangle)}}{I(\langle s, \langle s, s\rangle\rangle)}}{I(\langle s, \langle s, \langle s, s\rangle\rangle\rangle)}}{\ldots}$$

→ Hence ProVerif never resolves on $I(x)$, $I(y)$, ...

Global state in ProVerif

A small protocol

$$\begin{array}{lll} A & \rightarrow & \mathsf{enc}(s, \langle k_1, k_2 \rangle) \\ & & \mathsf{enc}(k_1, k) \\ & & \mathsf{enc}(k_2, k) \end{array}$$

$$\begin{array}{lll} B & \leftarrow & \mathsf{enc}(x, k) \\ & \rightarrow & x \ \textit{once} \end{array}$$

Global state in ProVerif

A small protocol

$$
\begin{array}{rcl}
A & \rightarrow & \mathrm{enc}(s, \langle k_1, k_2 \rangle) \\
 & & \mathrm{enc}(k_1, k) \\
 & & \mathrm{enc}(k_2, k)
\end{array}
$$

$$
\begin{array}{rcl}
B & \leftarrow & \mathrm{enc}(x, k) \\
 & \rightarrow & x \textit{ once}
\end{array}
$$

Horn clauses $\mathcal{C}$

$$
\begin{array}{rl}
\Rightarrow & I(\mathrm{enc}(s, \langle k_1, k_2 \rangle)) \\
\Rightarrow & I(\mathrm{enc}(k_1, k)) \\
\Rightarrow & I(\mathrm{enc}(k_2, k))
\end{array}
$$

$$
I(\mathrm{enc}(x, k)) \;\Rightarrow\; I(x)
$$

$$
\left.\begin{array}{rcl}
I(\mathrm{enc}(x, y)), I(y) & \Rightarrow & I(y) \\
I(\langle x, y \rangle) & \Rightarrow & I(x) \\
I(\langle x, y \rangle) & \Rightarrow & I(y)
\end{array}\right\} \begin{array}{l}\text{attacker} \\ \text{clauses}\end{array}
$$

s can be proved to remain secret if $\mathcal{C} \not\vdash I(s)$.

Global state in ProVerif

A small protocol

$$\begin{array}{rcl} A & \rightarrow & \mathrm{enc}(s, \langle k_1, k_2 \rangle) \\ & & \mathrm{enc}(k_1, k) \\ & & \mathrm{enc}(k_2, k) \end{array}$$

$$\begin{array}{rcl} B & \leftarrow & \mathrm{enc}(x, k) \\ & \rightarrow & x \textit{ once} \end{array}$$

Horn clauses $\mathcal{C}$

$$\begin{array}{rl} \Rightarrow & I(\mathrm{enc}(s, \langle k_1, k_2 \rangle)) \\ \Rightarrow & I(\mathrm{enc}(k_1, k)) \\ \Rightarrow & I(\mathrm{enc}(k_2, k)) \end{array}$$

$$I(\mathrm{enc}(x, k)) \Rightarrow I(x)$$

$$\left.\begin{array}{rcl} I(\mathrm{enc}(x, y)), I(y) & \Rightarrow & I(y) \\ I(\langle x, y \rangle) & \Rightarrow & I(x) \\ I(\langle x, y \rangle) & \Rightarrow & I(y) \end{array}\right\} \text{attacker clauses}$$

s can be proved to remain secret if $\mathcal{C} \nvdash I(s)$.

However, $\mathcal{C} \vdash I(s)$!

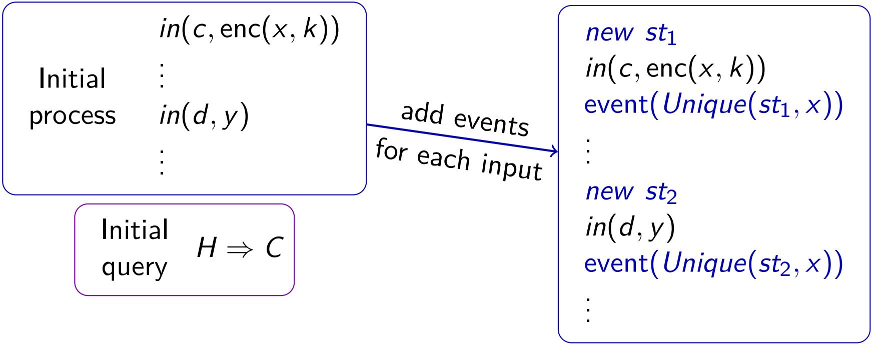
The idea

Initial process

$$\begin{array}{l} in(c, \mathrm{enc}(x, k)) \\ \vdots \\ in(d, y) \\ \vdots \end{array}$$

Initial query $H \Rightarrow C$

The idea



Initial process

$in(c, \mathrm{enc}(x, k))$

$\vdots$

$in(d, y)$

$\vdots$

Initial query $H \Rightarrow C$

add events for each input

$new\ st_1$

$in(c, \mathrm{enc}(x, k))$

$\mathrm{event}(Unique(st_1, x))$

$\vdots$

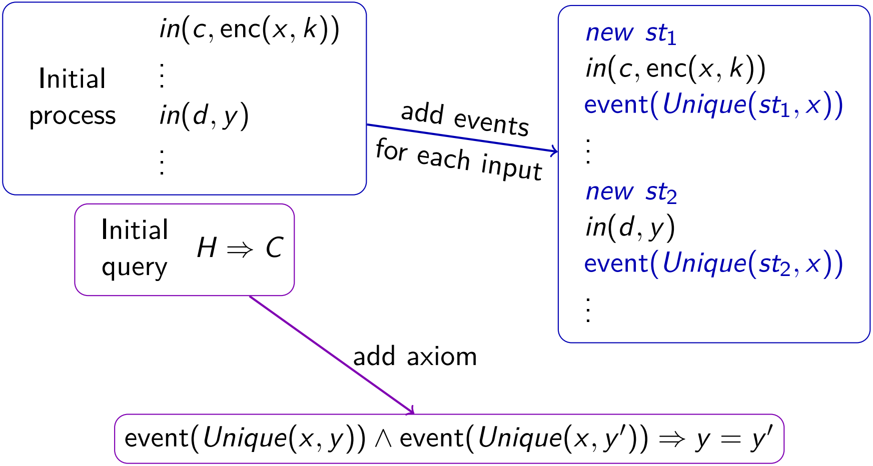
$new\ st_2$

$in(d, y)$

$\mathrm{event}(Unique(st_2, x))$

$\vdots$

The idea



Initial process

$in(c, \mathrm{enc}(x, k))$

$\vdots$

$in(d, y)$

$\vdots$

add events for each input

$new\ st_1$

$in(c, \mathrm{enc}(x, k))$

$\mathrm{event}(Unique(st_1, x))$

$\vdots$

$new\ st_2$

$in(d, y)$

$\mathrm{event}(Unique(st_2, x))$

$\vdots$

Initial query $H \Rightarrow C$

add axiom

$\mathrm{event}(Unique(x, y)) \wedge \mathrm{event}(Unique(x, y')) \Rightarrow y = y'$

Other limitations of ProVerif

1. Horn clauses yield over-aproximations
   Example: $\forall v \quad I(v) \Rightarrow \mathrm{Voted}(id, v, r(v))$

Other limitations of ProVerif

1. Horn clauses yield over-aproximations
   Example: $\forall v \quad I(v) \Rightarrow \mathrm{Voted}(id, v, r(v))$
   yields
   $\mathrm{Voted}(id, v_1, r(v_1)), \mathrm{Voted}(id, v_2, r(v_2)), \mathrm{Voted}(id, v_3, r(v_3))$
   $\ldots$

Other limitations of ProVerif

1. Horn clauses yield over-aproximations
   Example: $\forall v \quad I(v) \Rightarrow \mathrm{Voted}(id, v, r(v))$
   yields
   $\mathrm{Voted}(id, v_1, r(v_1)), \mathrm{Voted}(id, v_2, r(v_2)), \mathrm{Voted}(id, v_3, r(v_3))$
   ...

   Idea: restrictions

   $$\mathrm{Voted}(id, v_1, r_1), \mathrm{Voted}(id, v_2, r_2) \Rightarrow v_1 = v_2 \text{ AND } r_1 = r_2$$

Other limitations of ProVerif

1. Horn clauses yield over-aproximations
   Example: $\forall v \quad I(v) \Rightarrow \mathrm{Voted}(id, v, r(v))$
   yields
   $\mathrm{Voted}(id, v_1, r(v_1)), \mathrm{Voted}(id, v_2, r(v_2)), \mathrm{Voted}(id, v_3, r(v_3))$
   $\ldots$

   Idea: restrictions

   $$\mathrm{Voted}(id, v_1, r_1), \mathrm{Voted}(id, v_2, r_2) \Rightarrow v_1 = v_2 \text{ AND } r_1 = r_2$$

2. Saturation by resolution may still not terminate
   (despite ProVerif's strategy)

Other limitations of ProVerif

1. Horn clauses yield over-aproximations
   Example: $\forall v \quad I(v) \Rightarrow \text{Voted}(id, v, r(v))$
   yields
   $\text{Voted}(id, v_1, r(v_1)), \text{Voted}(id, v_2, r(v_2)), \text{Voted}(id, v_3, r(v_3))$
   $\ldots$

   Idea: restrictions

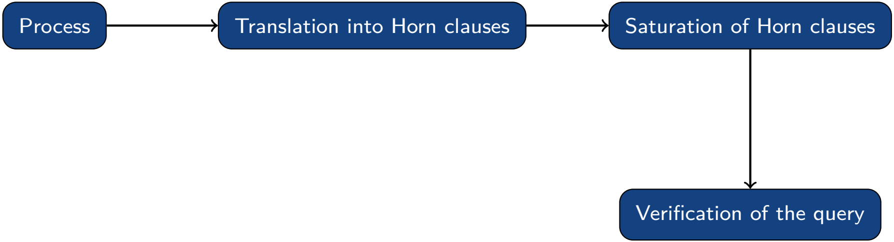
   $$\text{Voted}(id, v_1, r_1), \text{Voted}(id, v_2, r_2) \Rightarrow v_1 = v_2 \text{ AND } r_1 = r_2$$

2. Saturation by resolution may still not terminate
   (despite ProVerif's strategy)

   Idea: lemma as proof helpers

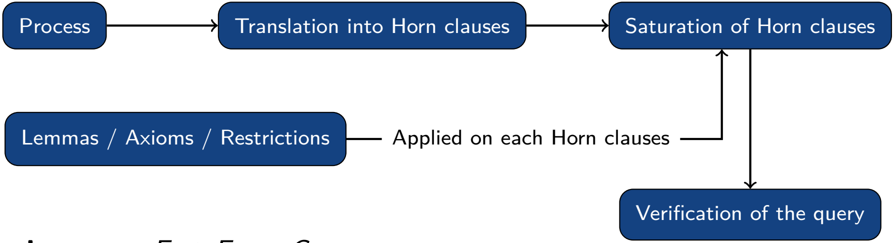
Proverif 2.02: introduction of lemmas

[S&P'22, with B. Blanchet and V. Cheval]


Process
Translation into Horn clauses
Saturation of Horn clauses
Verification of the query

Proverif 2.02: introduction of lemmas

[S&P'22, with B. Blanchet and V. Cheval]


Process
Translation into Horn clauses
Saturation of Horn clauses
Lemmas / Axioms / Restrictions
Applied on each Horn clauses
Verification of the query

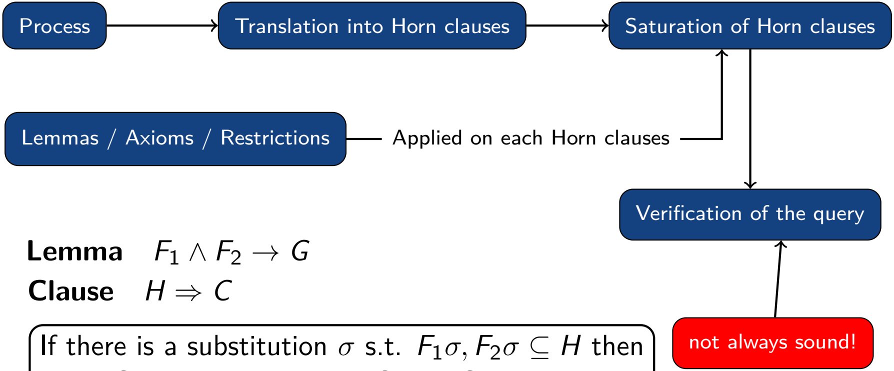
Lemma $F_1 \wedge F_2 \rightarrow G$

Clause $H \Rightarrow C$

If there is a substitution σ s.t. $F_1\sigma, F_2\sigma \subseteq H$ then $H \Rightarrow C$ is replaced by $H \wedge G\sigma \Rightarrow C$

Proverif 2.02: introduction of lemmas

[S&P'22, with B. Blanchet and V. Cheval]


Process
Translation into Horn clauses
Saturation of Horn clauses
Lemmas / Axioms / Restrictions
Applied on each Horn clauses
Verification of the query
not always sound!

Lemma $F_1 \wedge F_2 \rightarrow G$

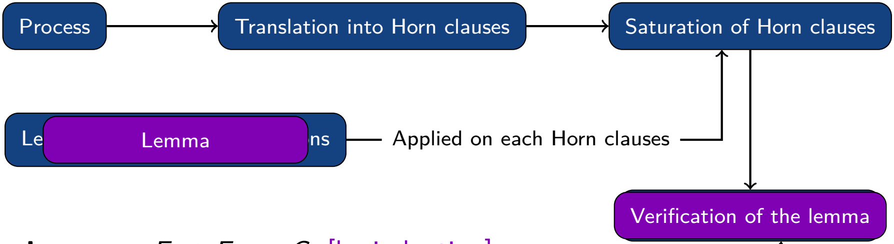
Clause $H \Rightarrow C$

If there is a substitution σ s.t. $F_1\sigma, F_2\sigma \subseteq H$ then $H \Rightarrow C$ is replaced by $H \wedge G\sigma \Rightarrow C$

events	✓
attacker facts	✗

Proverif 2.02: introduction of lemmas

[S&P'22, with B. Blanchet and V. Cheval]



Process

Translation into Horn clauses

Saturation of Horn clauses

Le

Lemma

ns

Applied on each Horn clauses

Verification of the lemma

Lemma $F_1 \wedge F_2 \rightarrow G$ [by induction]

Clause $H \Rightarrow C$

If there is a substitution σ s.t. $F_1\sigma, F_2\sigma \subseteq H$ then $H \Rightarrow C$ is replaced by $H \wedge G\sigma \Rightarrow C$

not always sound!

events ✓

attacker facts ✗

Even better: lemma by induction

Experimental results

Protocol	Q	Old	# queries	ProVerif 2.02
PCV Otway-Rees	eq	✗	1	✓
PCV Needham-Schroeder	inj	✗	6	✓
			3	↯
PCV Denning-Sacco	inj	✗	1	↯
JFK	cor	✗	2	↯
	inj		2	✓
Arinc823	cor	✗	6	↯
Helios-norevote	eq	✗	4	✓
Signal	cor	✗	2	↯
TLS12-TLS13-draft18	cor	✗	1	↯

Back to Belenios

	Who is dishonest?			
	∅	Serv	Reg	Serv+ Reg
Verifiability	✓	✓*	✓*	✗
recorded as cast	✓	✓*	✓*	✓*
tallied as recorded	✓	✓	✓	✓
eligibility verif.	✓	✓	✓	✗

(*) provided voters verify at the end of the election.

Back to Belenios

	Who is dishonest?			
	$\emptyset$	Serv	Reg	Serv+ Reg
Verifiability	✓	✓*	✓*	✗
recorded as cast	✓	✓*	✓*	✓*
tallied as recorded	✓	✓	✓	✓
eligibility verif.	✓	✓	✓	✗

(*) provided voters verify at the end of the election.

	Who is dishonest?	
	$\leq t$ trustees	$> t$ trustees
Vote privacy	✓	✗

Setting: the election key is shared amongst n trustees, $t+1$ trustees are needed to decrypt.

Back to Belenios

	Who is dishonest?			
	$\emptyset$	Serv	Reg	Serv+ Reg
Verifiability	✓	✓*	✓*	✗
recorded as cast	✓	✓*	✓*	✓*
tallied as recorded	✓	✓	✓	✓
eligibility verif.	✓	✓	✓	✗

(*) provided voters verify at the end of the election.

	Who is dishonest?	
	$\leq t$ trustees	$> t$ trustees
Vote privacy	✓	✗
in multi-elections	?	✗

Setting: the election key is shared amongst n trustees, $t+1$ trustees are needed to decrypt.

→ What about privacy in multi-election?

A closer look at privacy

Multi-elections:

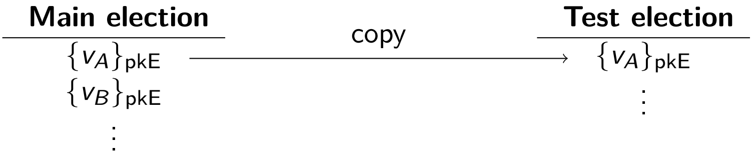
- elections with two rounds
- many elections at the same time (for different candidates)
- several elections circles ("voting stations")

Convenient feature: use the same key for all elections

- much easier for trustees
- In Belenios, voting credentials are refreshed for each election, avoiding confusion

A closer look at privacy (2)

Risk of key reuse: trustees used as decryption oracle



Main election

$\{v_A\}_{\text{pkE}}$

$\{v_B\}_{\text{pkE}}$

$\vdots$

copy

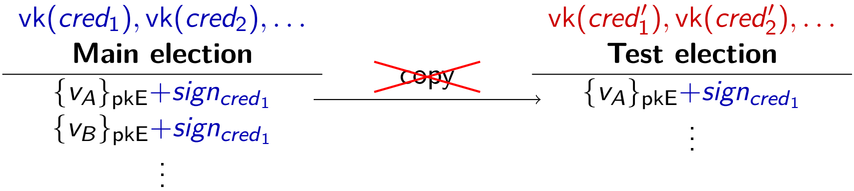
Test election

$\{v_A\}_{\text{pkE}}$

$\vdots$

A closer look at privacy (2)

Risk of key reuse: trustees used as decryption oracle



$\mathrm{vk}(cred_1), \mathrm{vk}(cred_2), \ldots$

Main election

$\{v_A\}_{\mathrm{pkE}} + sign_{cred_1}$

$\{v_B\}_{\mathrm{pkE}} + sign_{cred_1}$

$\vdots$

$\xrightarrow{\cancel{\text{copy}}}$

$\mathrm{vk}(cred'_1), \mathrm{vk}(cred'_2), \ldots$

Test election

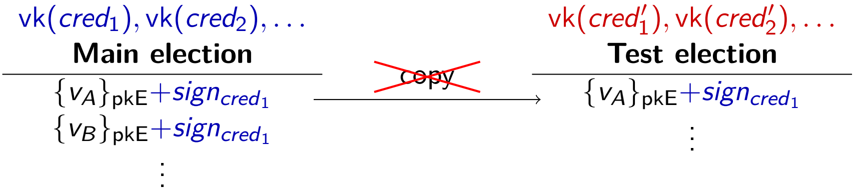
$\{v_A\}_{\mathrm{pkE}} + sign_{cred_1}$

$\vdots$

Not possible in Belenios since the *cred* are renewed.

A closer look at privacy (2)

Risk of key reuse: trustees used as decryption oracle



$\mathsf{vk}(cred_1), \mathsf{vk}(cred_2), \ldots$

Main election

$\{v_A\}_{\mathsf{pkE}} + sign_{cred_1}$

$\{v_B\}_{\mathsf{pkE}} + sign_{cred_1}$

$\vdots$

$\xrightarrow{\cancel{\text{copy}}}$

$\mathsf{vk}(cred'_1), \mathsf{vk}(cred'_2), \ldots$

Test election

$\{v_A\}_{\mathsf{pkE}} + sign_{cred_1}$

$\vdots$

Not possible in Belenios since the *cred* are renewed.

But, what if the Registrar is dishonest?
→ There is a flaw, fixed by chance: the server is a mandatory trustee, hence pk_E must be refreshed for each election
→ Require heavy monitoring in case both Registrar and Server are dishonest.

Ongoing detailed security model in Proverif

Limitations of Belenios

- No real booth
  → Internet voting IS remote voting
- Requires to trust the voter's computer
  A compromised computer could
  - leak the choice of the voter
  - change the vote for another candidate
    → Missing cast-as-intended
- Belenios is not "receipt free"
  → A voter can prove how they voted.

Some challenges

Better formal verification

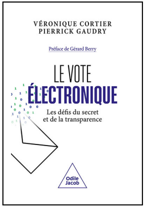
- decision procedures for larger equational theory classes
- better tools
- formalise security properties, possibly identifying new ones

Better e-voting systems

- more security properties: no vote buying, everlasting privacy, ...
- less trust assumptions (corrupted computers, ...)
- better authentication

Better involvement of the general public

- usability
- better legal regulation in many countries



VÉRONIQUE CORTIER
PIERRICK GAUDRY

Préface de Gérard Berry

LE VOTE
ÉLECTRONIQUE

Les défis du secret
et de la transparence

Odile Jacob