The quest for formally analyzing e-voting protocols

Steve Kremer

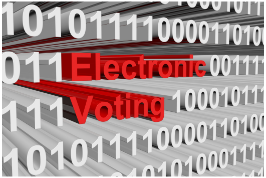
GT Méthodes Formelles pour la Sécurité

Cryptographic protocols everywhere!

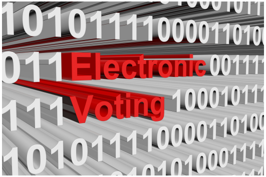
Distributed programs that

use crypto primitives (encryption, digital signature ,. . .)

to ensure security properties (confidentiality, authentication, anonymity,. . .)



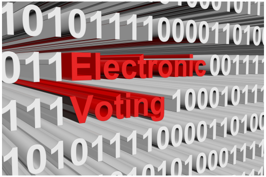
Cryptographic protocols are tricky!



Cryptographic protocols are tricky!

Bhargavan et al.:FREAK, Logjam, SLOTH, ...

Cremers et al., S&P'16

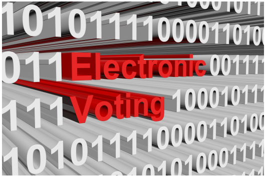


Cryptographic protocols are tricky!

Bhargavan et al.:FREAK, Logjam, SLOTH, . . .

Cremers et al., S&P'16

Arapinis et al., CCS'12

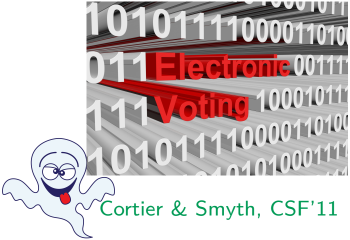


Cryptographic protocols are tricky!

Bhargavan et al.:FREAK, Logjam, SLOTH, . . .

Cremers et al., S&P'16

Arapinis et al., CCS'12



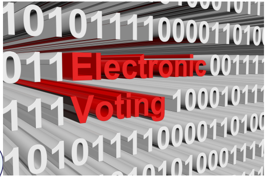
Cortier & Smyth, CSF'11

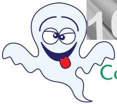
Cryptographic protocols are tricky!

Bhargavan et al.:FREAK, Logjam, SLOTH, . . .

Cremers et al., S&P'16

Arapinis et al., CCS'12





Cortier & Smyth, CSF'11

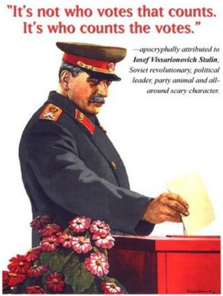
Steel et al., CSF'08, CCS'10

Electronic voting

Elections are a security-sensitive process which is the cornerstone of modern democracy

Electronic voting promises

- convenient, efficient and secure facility for recording and tallying votes
- for a variety of types of elections: from small committees or on-line communities through to full-scale national elections



Electronic voting

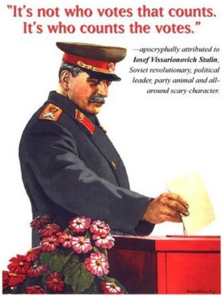
Elections are a security-sensitive process which is the cornerstone of modern democracy

Electronic voting promises

- convenient, efficient and secure facility for recording and tallying votes
- for a variety of types of elections: from small committees or on-line communities through to full-scale national elections

E-voting may include:

- use of voting machines in polling stations
- remote voting, via Internet (i-voting)



Real-world Internet elections

Recent political legally binding Internet elections in Europe:

- stepwise introduction in Switzerland (several cantons)
- parliamentary election in Estonia (all eligible voters)
- municipal and county elections in Norway (selected municipalities, selected voter groups)
- parliamentary elections in France (“expats”) in 2012

But also banned in Germany, Ireland, UK

Even more professional elections

Attacks!

Attacks by Alex Halderman and his team:

- attack on pilot project for overseas and military voters: took control of vote server, changed votes, removed root kit present on server, . . .
- Indian voting machines: clip-on memory manipulator
- Re-programmed e-voting machine used in US elections to play pack-man

Attacks!

Running PAC-MAN on a Sequaoia voting machine

Attacks!

Attacks by Alex Halderman and his team:

- attack on pilot project for overseas and military voters: took control of vote server, changed votes, removed root kit present on server, . . .
- Indian voting machines: clip-on memory manipulator
- Re-programmed e-voting machine used in US elections to play pack-man

. . . and many more

There exist also attacks on paper based remote voting, e.g. attack by Cortier *et al.* on a postal voting system used in CNRS elections

How can we avoid attacks?

How can we avoid attacks?

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Chancellerie fédérale ChF

Section des droits politiques

13 décembre 2013

Exigences techniques et administratives applicables au vote électronique

Entrée en vigueur: 15 janvier 2014

V. 1.0

How can we avoid attacks?

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Chancellerie fédérale ChF

Section des droits politiques

13 décembre 2013

Exigences techniques et administratives applicables au vote électronique

Entrée en vigueur: 15 janvier 2014

V. 1.0

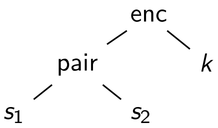
5.1. Contrôle du protocole cryptographique

5.1.1	Critères de contrôle: le protocole doit être conforme à l'objectif de sécurité et aux hypothèses de confiance figurant dans le modèle abstrait décrit au ch. 4. Pour cela, il doit exister une preuve cryptographique et une preuve symbolique. En ce qui concerne les composants cryptographiques fondamentaux, les preuves peuvent être apportées sur la base des hypothèses de sécurité généralement admises (par exemple « random oracle model », « decisional Diffie-Hellman assumption » et « Fiat-Shamir heuristic »). Le protocole doit se fonder si possible sur des protocoles éprouvés.

Symbolic models for protocol verification

Main ingredient of symbolic models

messages = **terms**



perfect cryptography (equational theories)

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x \quad \mathsf{fst}(\mathsf{pair}(x, y)) = x \quad \mathsf{snd}(\mathsf{pair}(x, y)) = y$$

the network **is** the attacker

- messages can be eavesdropped
- messages can be intercepted
- messages can be injected

Dolev, Yao: On the Security of Public Key Protocols. FOCS'81

Modelling the protocol

Protocols modelled in a process calculus, e.g. the applied pi calculus

$$
\begin{array}{rcll}
P ::= & & 0 & \\
& \mid & \mathrm{in}(c, x).P & \text{input} \\
& \mid & \mathrm{out}(c, t).P & \text{output} \\
& \mid & \text{if } t_1 = t_2 \text{ then } P \text{ else } Q & \text{conditional} \\
& \mid & P \parallel Q & \text{parallel} \\
& \mid & !P & \text{replication} \\
& \mid & \text{new } n.P & \text{restriction}
\end{array}
$$

Modelling the protocol

Protocols modelled in a process calculus, e.g. the applied pi calculus

$P ::=$		0	
	$\mid$	$\mathrm{in}(c, x).P$	input
	$\mid$	$\mathrm{out}(c, t).P$	output
	$\mid$	$\text{if } t_1 = t_2 \text{ then } P \text{ else } Q$	conditional
	$\mid$	$P \parallel Q$	parallel
	$\mid$	$!P$	replication
	$\mid$	$\text{new } n.P$	restriction

Specificities:

- messages are **terms** (not just names as in the pi calculus)
- equality in conditionals interpreted modulo an **equational theory**

Reasoning about attacker knowledge

Terms output by a process are organised in a **frame**:

$$\phi = \text{new } \bar{n}.\ \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$$

Reasoning about attacker knowledge

Terms output by a process are organised in a **frame**:

$$\phi = \text{new } \bar{n}.\ \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$$

Deducibility:

$\phi \vdash^{R} t$ if R is a public term and $R\phi =_E t$

Example

$$\varphi = \text{new } n_1, n_2, k_1, k_2.\ \{^{\text{enc}(n_1,k_1)}/_{x_1}, ^{\text{enc}(n_2,k_2)}/_{x_2}, ^{k_1}/_{x_3}\}$$

$$\varphi \vdash^{\text{dec}(x_1,x_3)} n_1 \qquad \varphi \nvdash n_2 \qquad \varphi \vdash^{\mathbf{1}} \mathbf{1}$$

Reasoning about attacker knowledge

Terms output by a process are organised in a **frame**:

$$\phi = \text{new } \bar{n}.\ \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$$

Static equivalence:

$\phi_1 \sim_s \phi_2$ if $\forall$ public terms R, R'.

$$R\phi_1 = R'\phi_1 \Leftrightarrow R\phi_2 = R'\phi_2$$

Examples

$$\text{new } k.\ \{^{\text{enc}(\mathbf{0},k)}/_{x_1}\} \sim_s \text{new } k.\ \{^{\text{enc}(\mathbf{1},k)}/_{x_1}\}$$

Reasoning about attacker knowledge

Terms output by a process are organised in a **frame**:

$$\phi = \text{new } \bar{n}.\ \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$$

Static equivalence:

$\phi_1 \sim_s \phi_2$ if $\forall$ public terms R, R'.

$$R\phi_1 = R'\phi_1 \Leftrightarrow R\phi_2 = R'\phi_2$$

Examples

$$\text{new } n_1, n_2.\ \{^{n_1}/_{x_1}, ^{n_2}/_{x_2}\} \not\sim_s \text{new } n_1, n_2.\ \{^{n_1}/_{x_1}, ^{n_1}/_{x_2}\}$$

Check $(x_1 \stackrel{?}{=} x_2)$

Reasoning about attacker knowledge

Terms output by a process are organised in a **frame**:

$$\phi = \text{new } \bar{n}.\ \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$$

Static equivalence:

$\phi_1 \sim_s \phi_2$ if $\forall$ public terms R, R'.

$$R\phi_1 = R'\phi_1 \Leftrightarrow R\phi_2 = R'\phi_2$$

Examples

$$\{^{\text{enc}(n,k)}/_{x_1}, ^{k}/_{x_2}\} \not\sim_s \{^{\text{enc}(\mathbf{0},k)}/_{x_1}, ^{k}/_{x_2}\}$$

Check $(dec(x_1, x_2) \stackrel{?}{=} \mathbf{0})$

From authentication to privacy

Many good tools:
AVISPA, Casper, Maude-NPA, ProVerif, Scyther, Tamarin, ...

Good at verifying **trace properties** (predicates on system behavior), e.g.,

- (weak) secrecy of a key
- authentication (correspondence properties)

  If B ended a session with A (and parameters p) then A must have started a session with B (and parameters p').

From authentication to privacy

Many good tools:
AVISPA, Casper, Maude-NPA, ProVerif, Scyther, Tamarin, . . .

Good at verifying **trace properties** (predicates on system behavior), e.g.,

- (weak) secrecy of a key
- authentication (correspondence properties)

If B ended a session with A (and parameters p) then A must have started a session with B (and parameters p').

Not all properties can be expressed on a trace.

$\rightsquigarrow$ recent interest in **indistinguishability properties**.

Indistinguishability as a process equivalence

Naturally modelled using **equivalences** from process calculi

Testing equivalence ($P \approx Q$)
for all processes A, we have that:

$$A \mid P \Downarrow c \text{ if, and only if, } A \mid Q \Downarrow c$$

$\longrightarrow$ $P \Downarrow c$ when P can send a message on the channel c.

Symbolic verification of e-voting protocols

What properties should an e-voting protocol satisfy?

How do we **model** these properties?

How van we **verify** these properties (automatically)?

What are the underlying **trust assumptions**?

Vote privacy

Anonymity of the vote:
no one should learn how I voted

Vote privacy

Anonymity of the vote:
no one should learn how I voted

We may want even more:

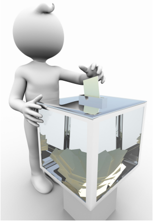
Receipt-freeness/coercion-resistance:
I cannot prove to someone else how I voted

$\rightsquigarrow$ avoid vote-buying / coercion

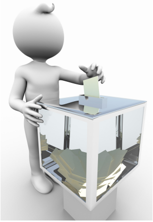
Election integrity through transparency

In traditional elections:

- transparent ballot box
- observers
- ...



Election integrity through transparency



In traditional elections:

- transparent ballot box
- observers
- ...

In e-voting: **End-to-end Verifiability**

- **Individual verifiability**: vote cast as intended
  e.g., voter checks his encrypted vote is on a public bulletin board
- **Universal verifiability**: vote counted as casted
  e.g., crypto proof that decryption was performed correctly
- **Eligibility verifiability**: only eligible votes counted
  e.g., crypto proof that every vote corresponds to a credential

$\leadsto$ **Verify the election, not the system!**

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

The attacker cannot learn the value of my vote

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

The attacker cannot learn the value of my vote

$\rightsquigarrow$ but the attacker knows values 0 and 1

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

~~The attacker cannot learn the value of my vote~~

The attacker cannot distinguish A votes and B votes:
$V_A(v) \approx V_B(v)$

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

~~The attacker cannot learn the value of my vote~~

The attacker cannot distinguish A votes and B votes:
$V_A(v) \approx V_B(v)$
$\rightsquigarrow$ but identities are revealed

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

~~The attacker cannot learn the value of my vote~~

~~The attacker cannot distinguish A votes and B votes:~~
~~$V_A(v) \approx V_B(v)$~~

The attacker cannot distinguish A votes 0 and A votes 1:
$V_A(0) \approx V_A(1)$

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

~~The attacker cannot learn the value of my vote~~

~~The attacker cannot distinguish A votes and B votes:~~
~~$V_A(v) \approx V_B(v)$~~

The attacker cannot distinguish A votes 0 and A votes 1:
$V_A(0) \approx V_A(1)$
$\rightsquigarrow$ but election outcome is revealed

How to model vote privacy?

How can we model

"**the attacker does not learn my vote (0 or 1)**"?

~~The attacker cannot learn the value of my vote~~

~~The attacker cannot distinguish A votes and B votes~~:
~~$V_A(v) \approx V_B(v)$~~

~~The attacker cannot distinguish A votes 0 and A votes 1~~:
~~$V_A(0) \approx V_A(1)$~~

The attacker cannot distinguish the situation where two honest voters swap votes:

$$V_A(0) \parallel V_B(1) \approx V_A(1) \parallel V_B(0)$$

K., Ryan: Analysis of an E-Voting Protocol in the Applied Pi Calculus. ESOP'05

Delaune, K., Ryan: Verifying privacy-type properties of e-voting protocols. JCS'09

The Helios e-voting protocol (MixNet version)

authenticated channel

BB

$V_1 \xrightarrow{\langle id_1, aenc(\mathsf{pk}_E, r_1, v_1)\rangle}$

$\langle id_1, \mathsf{aenc}(\mathsf{pk}_E, r_1, v_1)\rangle$

where pk_E is the election public key and MIX a verifiable mixnet.

The Helios e-voting protocol (MixNet version)

authenticated channel

$V_1 \xrightarrow{\langle id_1, aenc(\mathrm{pk}_E, r_1, v_1)\rangle}$

$V_2 \xrightarrow{\langle id_2, aenc(\mathrm{pk}_E, r_2, v_2)\rangle}$

BB

$\langle id_1, \mathrm{aenc}(\mathrm{pk}_E, r_1, v_1)\rangle$

$\langle id_2, aenc(\mathrm{pk}_E, r_2, v_2)\rangle$

where pk_E is the election public key and MIX a verifiable mixnet.

The Helios e-voting protocol (MixNet version)

authenticated channel

BB

$V_1 \xrightarrow{\langle id_1,\, aenc(\mathrm{pk}_E,\, r_1,\, v_1)\rangle}$ $\langle id_1, \mathrm{aenc}(\mathrm{pk}_E, r_1, v_1)\rangle$

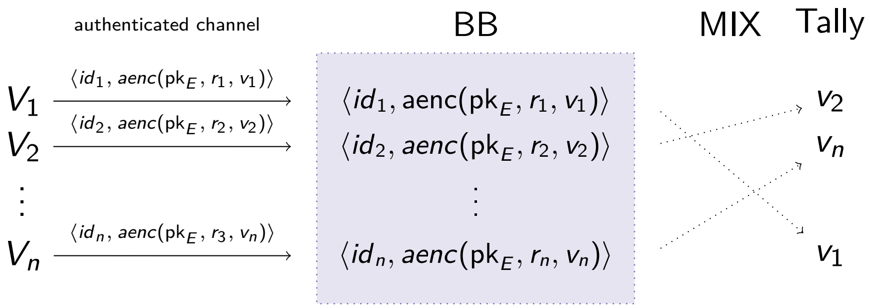
$V_2 \xrightarrow{\langle id_2,\, aenc(\mathrm{pk}_E,\, r_2,\, v_2)\rangle}$ $\langle id_2, aenc(\mathrm{pk}_E, r_2, v_2)\rangle$

$\vdots$ $\vdots$

$V_n \xrightarrow{\langle id_n,\, aenc(\mathrm{pk}_E,\, r_3,\, v_n)\rangle}$ $\langle id_n, aenc(\mathrm{pk}_E, r_n, v_n)\rangle$

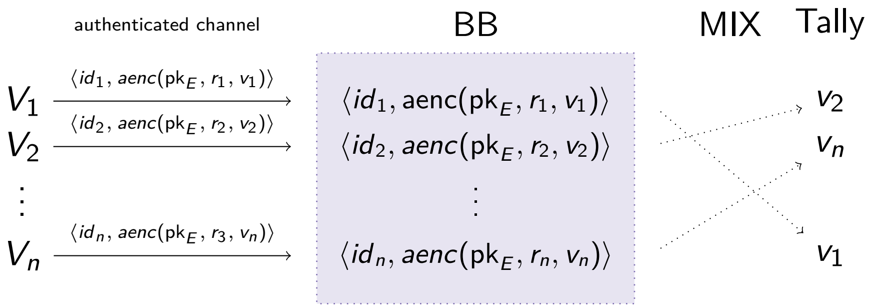
where pk_E is the election public key and MIX a verifiable mixnet.

The Helios e-voting protocol (MixNet version)



where pk_E is the election public key and MIX a verifiable mixnet.

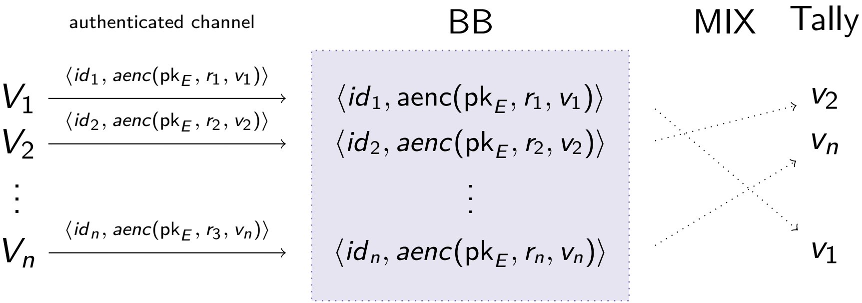
The Helios e-voting protocol (MixNet version)



where pk_E is the election public key and MIX a verifiable mixnet.

Privacy: $\mathrm{Helios}(v_1, v_2) \overset{?}{\approx}_t \mathrm{Helios}(v_2, v_1)$

The Helios e-voting protocol (MixNet version)

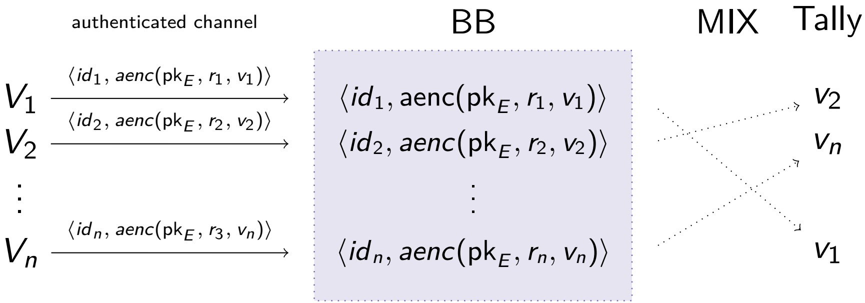


where pk_E is the election public key and MIX a verifiable mixnet.

Privacy: $\mathrm{Helios}(v_1, v_2) \overset{?}{\approx}_t \mathrm{Helios}(v_2, v_1)$ $\rightsquigarrow$ **replay attack!**

Cortier,Smyth: Attacking and Fixing Helios: An Analysis of Ballot Secrecy. CSF'11

The Helios e-voting protocol (MixNet version)



where pk_E is the election public key and MIX a verifiable mixnet.

Privacy: $\mathrm{Helios}(v_1, v_2) \overset{?}{\approx}_t \mathrm{Helios}(v_2, v_1)$ $\rightsquigarrow$ **replay attack!**

Fix: either use weeding, or zkp that voter knows encryption randomness

Automated verification

Which **scenario** should we analyse?

- How many honest/dishonest voters?
- Which authorities may be dishonest?
- Are voters allowed to revote? How many times?

Automated verification

Which **scenario** should we analyse?

- How many honest/dishonest voters?
- Which authorities may be dishonest?
- Are voters allowed to revote? How many times?

Which **tool** to use?

- verification of **equivalence properties**;
- many **crypto primitives**, zero-knowledge proofs, ideally homomorphic encryption;
- **private channels** useful for encoding possibility to revote

Automated verification

Which **scenario** should we analyse?

- How many honest/dishonest voters?
- Which authorities may be dishonest?
- Are voters allowed to revote? How many times?

Which **tool** to use?

- verification of **equivalence properties**;
- many **crypto primitives**, zero-knowledge proofs, ideally homomorphic encryption;
- **private channels** useful for encoding possibility to revote

All existing tools have some shortcomings.

3 Voters are enough!

For a "reasonable" class of e-voting protocols, for vote privacy
(including Helios, Belenios Civitas, Prêt-à-Voter,. . .)

It is sufficient to consider **3 voters** (2 honest + 1 dishonest).

When **no revote** is allowed **3 ballots** are sufficient.

When **revoting** is allowed, **10 ballots** are sufficient.

With **identifiable ballots**, **7 ballots** are sufficient.

3 Voters are enough!

For a “reasonable” class of e-voting protocols, for vote privacy
(including Helios, Belenios Civitas, Prêt-à-Voter,...)

It is sufficient to consider **3 voters** (2 honest + 1 dishonest).

When **no revote** is allowed **3 ballots** are sufficient.

When **revoting** is allowed, **10 ballots** are sufficient.

With **identifiable ballots**, **7 ballots** are sufficient.

Finite, but **large** number of scenarios!

Arapinis, Cortier, K.: *When are three voters are enough for privacy properties?*
ESORICS'16

DEEPSEC: DEciding Equivalence Properties in SECurity protocols

Decision procedure for trace equivalence
(no approximation, but high complexity coNEXP!)

Bounded number of sessions
(no replication; otherwise full applied pi)

Crypto primitives specified by
destructor subterm convergent rewrite systems

Tool implemented in OCaml:
`https://github.com/DeepSec-prover/deepsec`

Input language similar to (untyped) ProVerif

Possibility to distribute the verification
(on multiple cores and multiple machines)

Implements state-of-the art POR techniques

Cheval, K., Rakotonirina: *DEEPSEC: Deciding equivalence properties in security protocols – Theory and Practice* IEEE S&P'18

Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still **infinite**: unbounded number of attacker inputs!

Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still **infinite**: unbounded number of attacker inputs!

Idea: represent infinite number of possible inputs **symbolically** in a **constraint system**

Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still **infinite**: unbounded number of attacker inputs!

Idea: represent infinite number of possible inputs **symbolically** in a **constraint system**

Example

$\text{in}(c, x).P$ transitions to P but keeps a deduction constraint $X \vdash^? x$

Verification for a bounded number of sessions

Bounded number of sessions: why is it difficult?

The state space is still **infinite**: unbounded number of attacker inputs!

Idea: represent infinite number of possible inputs **symbolically** in a **constraint system**

Example

$\mathrm{in}(c, x).P$ transitions to P but keeps a deduction constraint $X \vdash^? x$

if $t_1 = t_2$ then P else Q : 2 transitions

to P with constraint $t_1 =^?_{\mathcal{R}} t_2$

to Q with constraint $t_1 \neq^?_{\mathcal{R}} t_2$

Constraint systems

A **constraint system** is a tuple $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$ where:

- $\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\}$ is a frame;
- D is a conjunction of deduction facts $X \vdash^? x$;
- E^1 is a conjunction of formulas $u =^?_{\mathcal{R}} v$ or $u \neq^?_{\mathcal{R}} v$.

A **solution** is a pair of substitutions Σ, σ such that

- $\Phi\sigma \vdash^{X\Sigma} x\sigma$ for all $X \vdash^? x \in \mathsf{D}$
- $u\sigma \bowtie v\sigma$ for all $u \bowtie v \in \mathsf{E}^1$

Note: Σ represents attacker inputs and constraints are such that it completely defines σ

Symbolic semantics

Symbolic semantics: associate a constraint system to the process (sample rules)

$$(\mathcal{P} \cup \{\!\{\text{if } u = v \text{ then } Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge u =^?_{\mathcal{R}} v))$$

$$(\mathcal{P} \cup \{\!\{\mathsf{in}(c, x).Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\mathsf{in}(c,X)}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi, \mathsf{D} \wedge X \vdash^? x, \mathsf{E}^1))$$

$$(\mathcal{P} \cup \{\!\{\mathsf{out}(c, t).Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\mathsf{out}(c,\mathsf{ax})}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi \cup \{\mathsf{ax} \mapsto t\}, \mathsf{D}, \mathsf{E}^1))$$

Symbolic semantics

Symbolic semantics: associate a constraint system to the process (sample rules)

$$(\mathcal{P} \cup \{\!\{\text{if } u = v \text{ then } Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge u =^?_{\mathcal{R}} v))$$

$$(\mathcal{P} \cup \{\!\{\mathsf{in}(c, x).Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\mathsf{in}(c,X)}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi, \mathsf{D} \wedge X \vdash^? x, \mathsf{E}^1))$$

$$(\mathcal{P} \cup \{\!\{\mathsf{out}(c, t).Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\mathsf{out}(c,\mathsf{ax})}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi \cup \{\mathsf{ax} \mapsto t\}, \mathsf{D}, \mathsf{E}^1))$$

Sound: if $(A, \mathcal{C}) \xrightarrow{\ell}_{\mathsf{s}} (A', \mathcal{C}')$ then for any $(\Sigma, \sigma) \in Sol(\mathcal{C})$ we have that $A\sigma \xrightarrow{\ell\Sigma} A'\sigma$

Complete: if $(\Sigma, \sigma) \in Sol(\mathcal{C})$ and $A\sigma \xrightarrow{\ell\Sigma} A'$ then $(A, \mathcal{C}) \xrightarrow{\ell}_{\mathsf{s}} (A', \mathcal{C}')$ and $\Sigma', \sigma' \in Sol(C')$ and $A''\sigma' = A'$

A simple example

$$P^b \triangleq \mathsf{in}(c, x).\,\mathsf{if}\ x = b\ \mathsf{then}\ \mathsf{out}(c, 0)\ \mathsf{else}\ \mathsf{out}(c, x) \qquad b \in \{0, 1\}$$

$$Q \triangleq \mathsf{in}(c, x).\mathsf{out}(c, x)$$

$P^0 \approx_t Q$ but $P^1 \not\approx_t Q$ (different behavior on input 1)

A simple example

$$P^b \triangleq \mathrm{in}(c,x).\,\mathrm{if}\ x = b\ \mathrm{then}\ \mathrm{out}(c,0)\ \mathrm{else}\ \mathrm{out}(c,x) \qquad b \in \{0,1\}$$

$$Q \triangleq \mathrm{in}(c,x).\mathrm{out}(c,x)$$

$P^0 \approx_t Q$ but $P^1 \not\approx_t Q$ (different behavior on input 1)

Symbolic transitions tree:

$$(\mathcal{P}_0^b, \mathcal{C}_\emptyset) \xrightarrow[\mathsf{s}]{\mathrm{in}(c,X)} (\mathcal{P}_1^b, \mathcal{C}_1^b) \xrightarrow[\mathsf{s}]{\varepsilon} (\mathcal{P}_2^b, \mathcal{C}_2^b) \xrightarrow[\mathsf{s}]{\mathrm{out}(c,\mathsf{ax}_1)} (\mathcal{P}_4^b, \mathcal{C}_4^b)$$

$$(\mathcal{P}_1^b, \mathcal{C}_1^b) \xrightarrow[\mathsf{s}]{\varepsilon} (\mathcal{P}_3^b, \mathcal{C}_3^b) \xrightarrow[\mathsf{s}]{\mathrm{out}(c,\mathsf{ax}_1)} (\mathcal{P}_5^b, \mathcal{C}_5^b)$$

$$(\mathcal{Q}_0, \mathcal{C}_\emptyset) \xrightarrow[\mathsf{s}]{\mathrm{in}(c,X)} (\mathcal{Q}_1, \mathcal{C}_1) \xrightarrow[\mathsf{s}]{\mathrm{out}(c,\mathsf{ax}_1)} (\mathcal{Q}_2, \mathcal{C}_2)$$

$$\begin{aligned}
\mathcal{C}_2 &\triangleq (\{\mathsf{ax}_1 \mapsto x\}, X \vdash^? x, \emptyset) \\
\mathcal{C}_4^b &\triangleq (\{\mathsf{ax}_1 \mapsto 0\}, X \vdash^? x, x =^?_{\mathcal{R}} b) \\
\mathcal{C}_4^b &\triangleq (\{\mathsf{ax}_1 \mapsto x\}, X \vdash^? x, x \neq^?_{\mathcal{R}} b)
\end{aligned}$$

Partition Tree

Build a **joint** symbolic execution tree

Partition solutions (split nodes): ensure static equivalences of all solutions in a same node
⇝ done by **constraint solving algorithm**

$(\mathcal{Q}_0, \mathcal{C}_\emptyset)$
$(\mathcal{P}_0^{\mathbf{0}}, \mathcal{C}_\emptyset)$

Partition Tree

Build a **joint** symbolic execution tree

Partition solutions (split nodes): ensure static equivalences of all solutions in a same node
$\rightsquigarrow$ done by **constraint solving algorithm**

$$\begin{matrix}(\mathcal{Q}_0, \mathcal{C}_\emptyset) \\ (\mathcal{P}_0^{\mathbf{0}}, \mathcal{C}_\emptyset)\end{matrix} \xRightarrow{\mathsf{in}(c,X)}_{\mathsf{s}} \begin{matrix}(\mathcal{Q}_1, \mathcal{C}_1),\ (\mathcal{P}_1^{\mathbf{0}}, \mathcal{C}_1^{\mathbf{0}}) \\ (\mathcal{P}_2^{\mathbf{0}}, \mathcal{C}_2^{\mathbf{0}}),\ (\mathcal{P}_3^{\mathbf{0}}, \mathcal{C}_3^{\mathbf{0}})\end{matrix}$$

Partition Tree

Build a **joint** symbolic execution tree

Partition solutions (split nodes): ensure static equivalences of all solutions in a same node
$\rightsquigarrow$ done by **constraint solving algorithm**

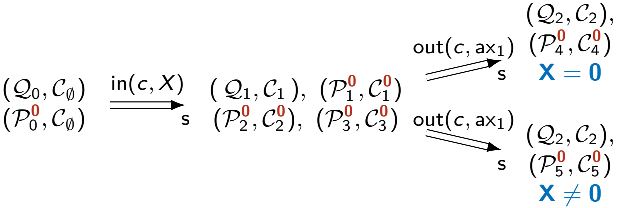
$$\begin{matrix}(\mathcal{Q}_0, \mathcal{C}_\emptyset) \\ (\mathcal{P}_0^{\mathbf{0}}, \mathcal{C}_\emptyset)\end{matrix} \xRightarrow{\mathsf{in}(c, X)}_{\mathsf{s}} \begin{matrix}(\mathcal{Q}_1, \mathcal{C}_1),\ (\mathcal{P}_1^{\mathbf{0}}, \mathcal{C}_1^{\mathbf{0}}) \\ (\mathcal{P}_2^{\mathbf{0}}, \mathcal{C}_2^{\mathbf{0}}),\ (\mathcal{P}_3^{\mathbf{0}}, \mathcal{C}_3^{\mathbf{0}})\end{matrix} \xRightarrow{\mathsf{out}(c, \mathsf{ax}_1)}_{\mathsf{s}} \begin{matrix}(\mathcal{Q}_2, \mathcal{C}_2), \\ (\mathcal{P}_4^{\mathbf{0}}, \mathcal{C}_4^{\mathbf{0}}),\ (\mathcal{P}_5^{\mathbf{0}}, \mathcal{C}_5^{0})\end{matrix}$$

Need to **partition**: $\mathcal{C}_4^{\mathbf{0}}$ enforces $X = 0$ and $\mathcal{C}_5^{\mathbf{0}}$ enforces $X \neq 0$.

Partition Tree

Build a **joint** symbolic execution tree

Partition solutions (split nodes): ensure static equivalences of all solutions in a same node
$\rightsquigarrow$ done by **constraint solving algorithm**

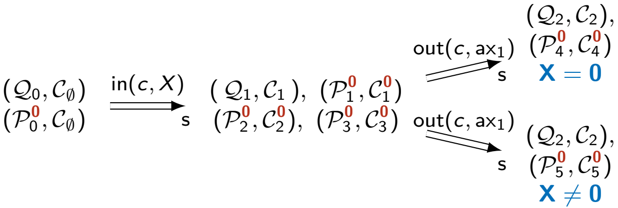


Need to **partition**: $\mathcal{C}^{\mathbf{0}}_4$ enforces $X = 0$ and $\mathcal{C}^{\mathbf{0}}_5$ enforces $X \neq 0$.

Partition Tree

Build a **joint** symbolic execution tree

Partition solutions (split nodes): ensure static equivalences of all solutions in a same node
$\rightsquigarrow$ done by **constraint solving algorithm**

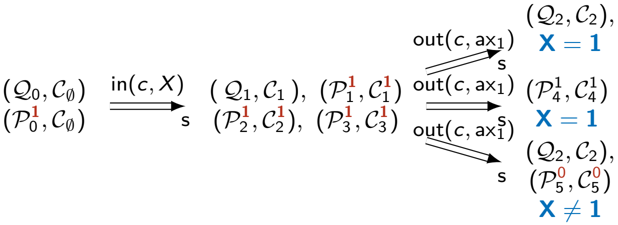


Need to **partition**: $\mathcal{C}^{\mathbf{0}}_4$ enforces $X = 0$ and $\mathcal{C}^{\mathbf{0}}_5$ enforces $X \neq 0$.
$\mathbf{P^0 \approx_t Q}$: each leaf contains processes derived from $P^{\mathbf{0}}$ and Q.

Partition Tree

Build a **joint** symbolic execution tree

Partition solutions (split nodes): ensure static equivalences of all solutions in a same node
⇝ done by **constraint solving algorithm**



Need to **partition more** to ensure static equivalence inside nodes.
$\mathbf{P^1 \not\approx_t Q}$: leaves with processes only from P^1.

DEEPSEC in practice

Verifying strong secrecy in classical authentication protocols

Protocol (# of roles)		Akiss	APTE	SPEC	Sat-Eq	DeepSec
Denning-Sacco	3	✓ <1s	✓ <1s	✓ 11s	✓ <1s	✓ <1s
	6	✓ <1s	✓ 1s	OM	✓ <1s	✓ <1s
	7	✓ 6s	✓ 3s		✓ <1s	✓ <1s
	10	OM	✓ 9m49		✓ <1s	✓ <1s
	12		⏱		✓ <1s	✓ <1s
	29				✓ <1s	✓ 6s
Yahalom-Lowe	3	✓ <1s	✓ <1s	✓ 7s	✓ <1s	✓ <1s
	6	✓ 2s	✓ 41s	OM	✓ <1s	✓ <1s
	7	✓ 42s	✓ 34m38s		✓ 1s	✓ <1s
	10	OM	⏱		✓ 1s	✓ <1s
	17				✓ 12s	✓ 8s
Otway-Rees	3	✓ 28s	✓ 2s	✓ 58m9s	✗	✓ <1s
	6	OM	OM	⏱		✓ <1s
	7					✓ <1s
	14					✓ 5m28s

✓ equivalence proved ✗ out of scope

OM out of memory/stack overflow ⏱ timeout (12H)

DEEPSEC in practice

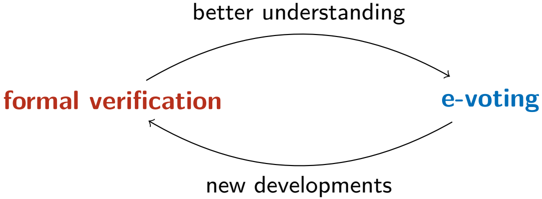
Verifying vote privacy on different versions of Helios

Helios variant (# roles)		DeepSec
Vanilla	6	↯ <1s
No revote Weeding	6	✓ 1s
No revote ZKP	6	✓ 2s
Dishonest revote Weeding	10	✓ 30m 24s
Dishonest revote ZKP	10	✓ 9m 26s
Honest revote Weeding	11	↯ 2s
Honest revote ZKP	11	✓ 2h 42m

Honest revote {Weeding|ZKP} means
1 honest voter revotes; 7 ballots accepted.

Several honest revotes still out-of-scope because of state explosion.

Conclusion



State explosion: more general POR techniques in DEEPSEC may enable verification of "full scenario".

Nearly no work on verifiability. Still need for good definitions that can be automatically verified.

E-voting on dishonest platforms: increases attacker power and complicates the protocol.