Selene in Celf: Formalising Voting Protocols in Linear Logic

Alessandro Bruni
(joint work with Carsten Schürmann)

brun@itu.dk

University of Luxembourg

DemTech
Democracy, Technology & Trust

IT UNIVERSITY OF COPENHAGEN

الصندوق القطري لرعاية البحث العلمي
Qatar National Research Fund
Member of Qatar Foundation عضو في مؤسسة قطر

Abstract

Designing security protocols is a task notoriously known to be prone to mistakes. Voting protocols in particular have subtle and often contrasting properties like vote verifiability, receipt-freenes and coercion resistance, hence their precise characterisation is often complex to understand. In this talk, we aim to show how the foundational framework of linear logic can help to produce clear specifications of complex protocols, using the Selene internet voting protocol as a case in point. The notions of coherence and concurrency built into the linear logical framework Celf provide an initial check on the protocol well-formedness, and more advanced security properties can be expressed using dependent types and the higher-order syntactic approach of Celf.

Internet Elections — Challenges

- Election Integrity
- Ballot Secrecy
- Transparency
- Security
- Coercibility
- Receipts

German Supreme Court

Law permitting the use of electronic election machines is unconstitutional.

[Senat 2 BvC 3/07]

Universal Declaration of Human Rights

The will of the people shall be the basis of the authority of government; this will shall be expressed in periodic and genuine elections which shall be by universal and equal suffrage and shall be held by secret vote or by equivalent free voting procedures.

[Article 21.3]

How to design a Voting Protocol

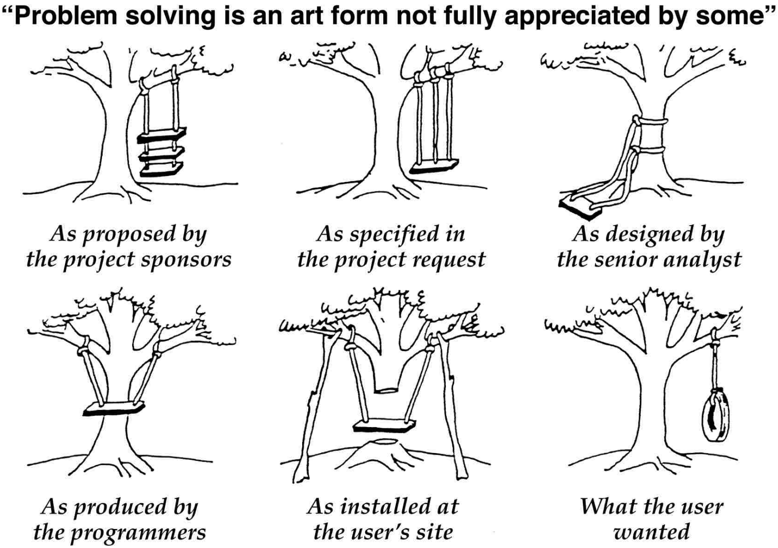
Requirements (highly country dependent)

- Requirements (country dependent)
- Single Points of Failures
- Operational protocols
- Verifiability

Design of a voting protocol

- Cryptographic techniques
- Evidence production
- Individual Verifiability
- Universal Verifiability

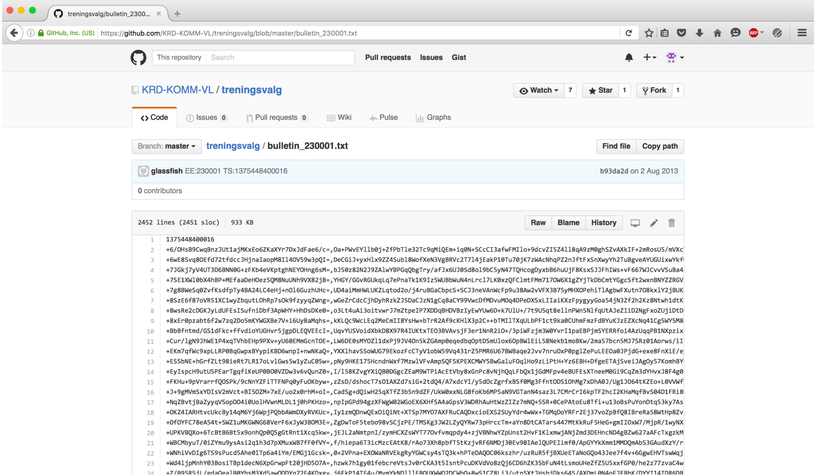
Designing a Protocol: Academia vs. Reality



Tree Swing graphic by S Høgh 1993 - from Businessballs.com/treeswing.htm 2013

The Norwegian Ballot Decryption Ceremony

Zero Knowledge Proofs of Knowledge *pfk*, *pfk'*



The Selene E-voting Protocol

- Need to present?

The Selene E-voting Protocol

- Need to present!

The Selene E-voting Protocol

- Need to present!
- Due to Peter Ryan, Peter Rønne, Vincenzo Iovino

The Selene E-voting Protocol

- Need to present!
- Due to Peter Ryan, Peter Rønne, Vincenzo Iovino
  dear audience. . . :)

The Selene E-voting Protocol

- Need to present!
- Due to Peter Ryan, Peter Rønne, Vincenzo Iovino
  dear audience. . . :)
- Internet voting protocol designed low-coercion scenarios

The Selene E-voting Protocol

- Need to present!
- Due to Peter Ryan, Peter Rønne, Vincenzo Iovino
  dear audience. . . :)
- Internet voting protocol designed low-coercion scenarios

Key ideas

1. votes are publicly posted on a bullettin board makes it easy to trust the result;
2. tracking receipts (*tracker numbers*) allow users to trust that their vote has been cast,
   individual verifiability
3. and to fake receipts for potential coercers.
   receipt freeness

El-gamal cryptosystem

Gen: Select a subgroup $G \subset \mathbb{Z}_p^*$ of order q, and a generator g of G. Choose $x \underset{R}{\leftarrow} Z_q$. Reveal $h = g^x$.

Enc: To encrypt a message $m \in G$, we choose $r \underset{R}{\leftarrow} Z_q$. The ciphertext is then:

$$(c, d) = (g^r, m \cdot h^r).$$

Dec: To decrypt the ciphertext (c, d), compute

$$m = \frac{d}{c^x}.$$

El-gamal homomorphisms:

Reencryption: $(g^r, m \cdot h^r)$, choose $r' \underset{R}{\leftarrow} Z_q$. Then $(g^{r+r'}, m \cdot h^{r+r'}) = (g^r, m \cdot h^r) \cdot (g^{r'}, 1 \cdot h^{r'})$ is a reencryption of m.

El-gamal homomorphisms:

Reencryption: $(g^r, m \cdot h^r)$, choose $r' \underset{R}{\leftarrow} Z_q$. Then $(g^{r+r'}, m \cdot h^{r+r'}) = (g^r, m \cdot h^r) \cdot (g^{r'}, 1 \cdot h^{r'})$ is a reencryption of m.

Additive homomorphism Let:

$$(c_1, d_1) = (g^{r_1}, g^{m_1} \cdot h^{r_1}) \qquad (c_2, d_2) = (g^{r_2}, g^{m_2} \cdot h^{r_2})$$

then

$$(c_1 \cdot c_2, d_1 \cdot d_2) = (g^{r_1+r_2}, g^{m_1+m_2} \cdot h^{r_1+r_2})$$

computes the sum of m_1 and m_2 under El-gamal using public key $h = g^x$.

El-gamal homomorphisms:

Reencryption: $(g^r, m \cdot h^r)$, choose $r' \underset{R}{\leftarrow} Z_q$. Then $(g^{r+r'}, m \cdot h^{r+r'}) = (g^r, m \cdot h^r) \cdot (g^{r'}, 1 \cdot h^{r'})$ is a reencryption of m.

Additive homomorphism Let:

$$(c_1, d_1) = (g^{r_1}, g^{m_1} \cdot h^{r_1}) \qquad (c_2, d_2) = (g^{r_2}, g^{m_2} \cdot h^{r_2})$$

then

$$(c_1 \cdot c_2, d_1 \cdot d_2) = (g^{r_1+r_2}, g^{m_1+m_2} \cdot h^{r_1+r_2})$$

computes the sum of m_1 and m_2 under El-gamal using public key $h = g^x$.
Note: if $m_1 + m_2$ is not to big, it is possible to solve efficiently the discrete logarithm of $g^{m_1+m_2}$ to obtain the sum.

Pedersen commitment

Gen: Select a subgroup $G \subset \mathbb{Z}_p^*$ of order q, and a generator g of G. Choose $x \underset{R}{\leftarrow} Z_q$. Reveal $h = g^x$.

Pedersen commitment

Gen: Select a subgroup $G \subset \mathbb{Z}_p^*$ of order q, and a generator g of G. Choose $x \underset{R}{\leftarrow} Z_q$. Reveal $h = g^x$.

Commit: To commit to a message $m \in G$, we choose $r \underset{R}{\leftarrow} Z_q$. The commitment is then: $c = g^m \cdot h^r$.

Pedersen commitment

Gen: Select a subgroup $G \subset \mathbb{Z}_p^*$ of order q, and a generator g of G. Choose $x \underset{R}{\leftarrow} Z_q$. Reveal $h = g^x$.

Commit: To commit to a message $m \in G$, we choose $r \underset{R}{\leftarrow} Z_q$. The commitment is then: $c = g^m \cdot h^r$.

Open: To reveal the message m, the second component is sent: $d = g^r$.

Pedersen commitment

Gen: Select a subgroup $G \subset \mathbb{Z}_p^*$ of order q, and a generator g of G. Choose $x \underset{R}{\leftarrow} Z_q$. Reveal $h = g^x$.

Commit: To commit to a message $m \in G$, we choose $r \underset{R}{\leftarrow} Z_q$. The commitment is then: $c = g^m \cdot h^r$.

Open: To reveal the message m, the second component is sent: $d = g^r$.

Properties

Information theoretically hiding: given the commitment c, any message $m' \in G$ is equally likely, and in particular, having the secret key x one can compute: $r' = \frac{m-m'}{x} + r$

Pedersen commitment

Gen: Select a subgroup $G \subset \mathbb{Z}_p^*$ of order q, and a generator g of G. Choose $x \underset{R}{\leftarrow} Z_q$. Reveal $h = g^x$.

Commit: To commit to a message $m \in G$, we choose $r \underset{R}{\leftarrow} Z_q$. The commitment is then: $c = g^m \cdot h^r$.

Open: To reveal the message m, the second component is sent: $d = g^r$.

Properties

Information theoretically hiding: given the commitment c, any message $m' \in G$ is equally likely, and in particular, having the secret key x one can compute: $r' = \frac{m-m'}{x} + r$

Computationally binding: finding two messages m and m' that open the commitment c requires finding an r and r' s.t. $g^m \cdot h^r = g^{m'} \cdot h^{r'}$; then one can compute $\log_g(h) = \frac{m'-m}{r-r'}$.

Overview

Actors

1. Election Authority (EA)
2. Web Bulletin Board (WBB)
3. Mixnet (M)
4. Teller(s) (T)
5. Voters (V_i)

Voting in seven easy steps

1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;

Voting in seven easy steps

1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e'_i that loses connection to n_i;

Voting in seven easy steps

1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e'_i that loses connection to n_i;
3. Teller(s) decrypt e_is, assign them to Voters V_i and generate Pedersen commitments c_i, then publish them to the Bulletin Board

Voting in seven easy steps

1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e_i' that loses connection to n_i;
3. Teller(s) decrypt e_is, assign them to Voters V_i and generate Pedersen commitments c_i, then publish them to the Bulletin Board
4. Votes v_i are encrypted (ev_i) and signed (s_i) by Voters V_i, and published along

Voting in seven easy steps

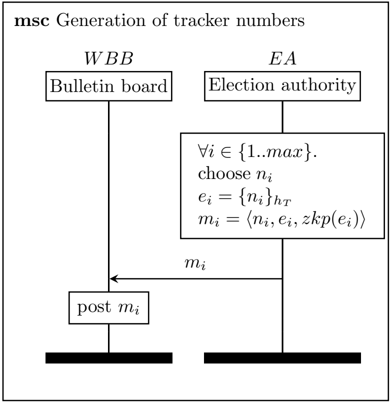
1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e'_i that loses connection to n_i;
3. Teller(s) decrypt e_is, assign them to Voters V_i and generate Pedersen commitments c_i, then publish them to the Bulletin Board
4. Votes v_i are encrypted (ev_i) and signed (s_i) by Voters V_i, and published along
5. Encrypted tracking numbers and votes $\langle e'_i, ev_i \rangle$ are shuffled by the Mixnet, then published as $\langle e''_i, ev'_i \rangle$, losing link to the originals;

Voting in seven easy steps

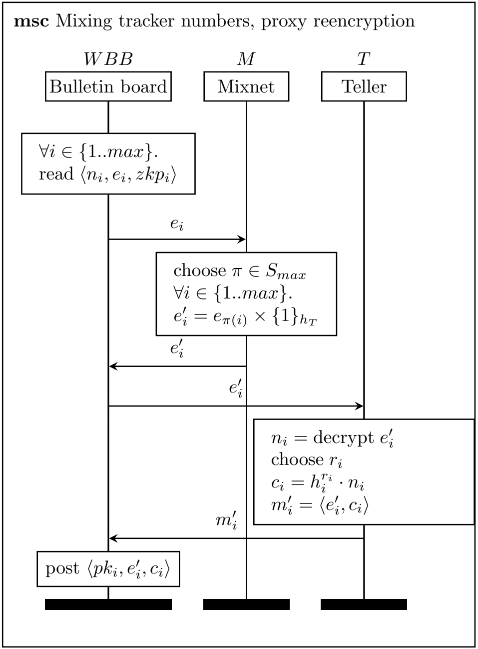
1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e'_i that loses connection to n_i;
3. Teller(s) decrypt e_is, assign them to Voters V_i and generate Pedersen commitments c_i, then publish them to the Bulletin Board
4. Votes v_i are encrypted (ev_i) and signed (s_i) by Voters V_i, and published along
5. Encrypted tracking numbers and votes $\langle e'_i, ev_i \rangle$ are shuffled by the Mixnet, then published as $\langle e''_i, ev'_i \rangle$, losing link to the originals;
6. Votes ev_i are decrypted by the Tellers, and published to the Bulletin Board

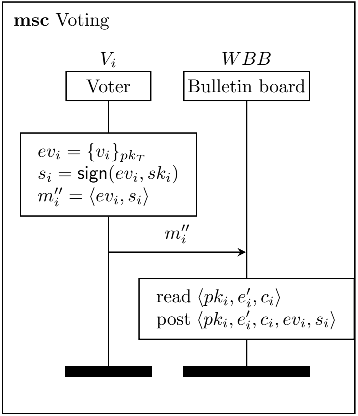
Voting in seven easy steps

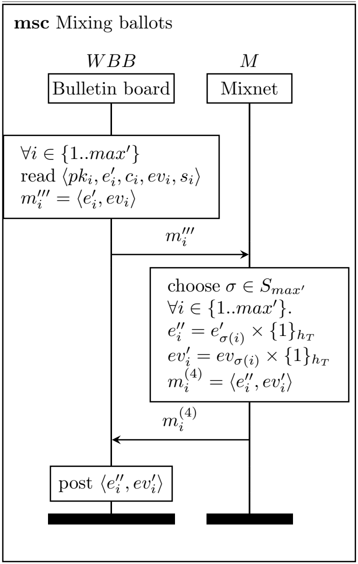
1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e_i' that loses connection to n_i;
3. Teller(s) decrypt e_is, assign them to Voters V_i and generate Pedersen commitments c_i, then publish them to the Bulletin Board
4. Votes v_i are encrypted (ev_i) and signed (s_i) by Voters V_i, and published along
5. Encrypted tracking numbers and votes $\langle e_i', ev_i \rangle$ are shuffled by the Mixnet, then published as $\langle e_i'', ev_i' \rangle$, losing link to the originals;
6. Votes ev_i are decrypted by the Tellers, and published to the Bulletin Board
7. Commitments are revealed by the Teller(s) to the Voters, who can check that their vote has been casted.



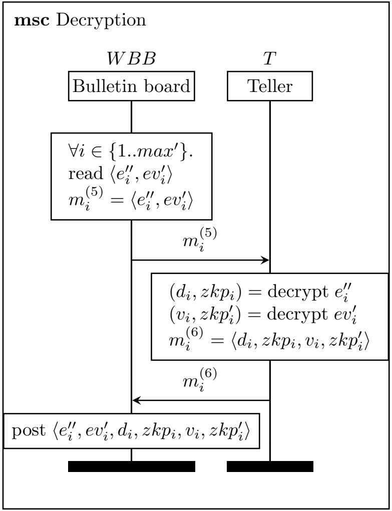
msc Generation of tracker numbers
WBB
Bulletin board
EA
Election authority
$\forall i \in \{1..max\}$.
choose n_i
$e_i = \{n_i\}_{h_T}$
$m_i = \langle n_i, e_i, zkp(e_i) \rangle$
m_i
post m_i

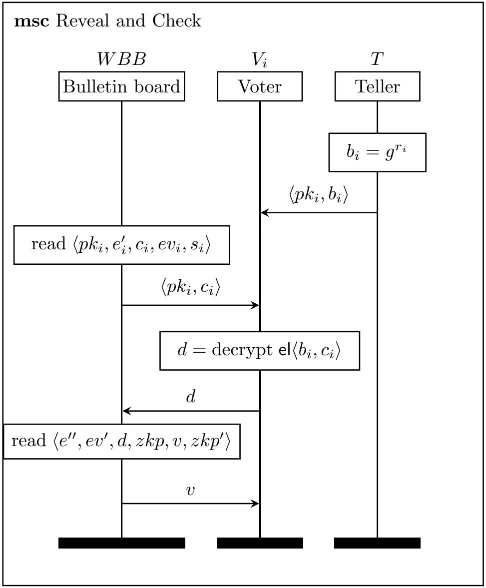

msc Mixing tracker numbers, proxy reencryption
WBB
Bulletin board
M
Mixnet
T
Teller
$\forall i \in \{1..max\}$.
read $\langle n_i, e_i, zkp_i \rangle$
e_i
choose $\pi \in S_{max}$
$\forall i \in \{1..max\}$.
$e'_i = e_{\pi(i)} \times \{1\}_{h_T}$
e'_i
e'_i
$n_i =$ decrypt e'_i
choose r_i
$c_i = h_i^{r_i} \cdot n_i$
$m'_i = \langle e'_i, c_i \rangle$
m'_i
post $\langle pk_i, e'_i, c_i \rangle$


msc Voting
V_i
WBB
Voter
Bulletin board
$ev_i = \{v_i\}_{pk_T}$
$s_i = \mathsf{sign}(ev_i, sk_i)$
$m''_i = \langle ev_i, s_i \rangle$
m''_i
read $\langle pk_i, e'_i, c_i \rangle$
post $\langle pk_i, e'_i, c_i, ev_i, s_i \rangle$



msc Mixing ballots
WBB
Bulletin board
M
Mixnet
$\forall i \in \{1..max'\}$
read $\langle pk_i, e'_i, c_i, ev_i, s_i \rangle$
$m'''_i = \langle e'_i, ev_i \rangle$
m'''_i
choose $\sigma \in S_{max'}$
$\forall i \in \{1..max'\}$.
$e''_i = e'_{\sigma(i)} \times \{1\}_{h_T}$
$ev'_i = ev_{\sigma(i)} \times \{1\}_{h_T}$
$m^{(4)}_i = \langle e''_i, ev'_i \rangle$
$m^{(4)}_i$
post $\langle e''_i, ev'_i \rangle$


msc Decryption
WBB
Bulletin board
T
Teller
∀i ∈ {1..max′}.
read ⟨e″_i, ev′_i⟩
m_i^(5) = ⟨e″_i, ev′_i⟩
m_i^(5)
(d_i, zkp_i) = decrypt e″_i
(v_i, zkp′_i) = decrypt ev′_i
m_i^(6) = ⟨d_i, zkp_i, v_i, zkp′_i⟩
m_i^(6)
post ⟨e″_i, ev′_i, d_i, zkp_i, v_i, zkp′_i⟩


msc Reveal and Check
WBB
Bulletin board
V_i
Voter
T
Teller
$b_i = g^{r_i}$
$\langle pk_i, b_i \rangle$
read $\langle pk_i, e'_i, c_i, ev_i, s_i \rangle$
$\langle pk_i, c_i \rangle$
$d =$ decrypt el$\langle b_i, c_i \rangle$
d
read $\langle e'', ev', d, zkp, v, zkp' \rangle$
v

Tools

Proof Assistants

- Coq [Herberlin et al]
- Certicrypt, Easycrypt [Barthe et al]
- CryptoAgda [Gustafsson, Pouillard]
- Maude, Maude-NPA [Meseguer et al]
- Tamarin [Meier, et al]

Protocol Verifiers

- Applied Pi [Abadi et al]
- ProVerif [Blanchet et al]
- SetPI [Bruni, Mödersheim]
- NRL Analyzer [Meadows]

Table of Contents

Table of Contents

Linear Logic

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.

Linear Logic

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of food.

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

Let's specify a voter check-in process:

- *Consume* an authorization card to prevent multiple check-ins.

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

Let's specify a voter check-in process:

- *Consume* an authorization card to prevent multiple check-ins.
- Use linear implication, $A \multimap \{B\}$.
  - $A \multimap \{B\} \approx$ "*consume* resource A to produce B."

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

Let's specify a voter check-in process:

- *Consume* an authorization card to prevent multiple check-ins.
- Use linear implication, $A \multimap \{B\}$.
  - $A \multimap \{B\} \approx$ "*consume* resource A to produce B."

$$\textit{voting-auth-card} \multimap \{\textit{blank-ballot}\}$$

"If I *give* an authorization card, then I *get* a blank ballot."

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

Let's specify a voter check-in process:

- *Consume* an authorization card to prevent multiple check-ins.
- Use linear implication, $A \multimap \{B\}$.
  - $A \multimap \{B\} \approx$ "*consume* resource A to produce B."

$$\textit{voting-auth-card} \multimap \{\textit{blank-ballot}\}$$

"If I *give* an authorization card, then I *get* a blank ballot."

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

Let's specify a voter check-in process:

- *Consume* an authorization card to prevent multiple check-ins.
- Use linear implication, $A \multimap \{B\}$.
  - $A \multimap \{B\} \approx$ "*consume* resource A to produce B."

$$\textit{voting-auth-card} \multimap \{\textit{blank-ballot}\}$$

"If I *give* an authorization card, then I *get* a blank ballot."

What is linear logic?

Traditional Logic

"Truth is free."

- Assumptions may be used any number of times.
- The logic of facts.

Linear Logic

"Truth is a consumable resource."

- Assumptions must be used exactly once.
- The logic of ~~food~~ voting.

Let's specify a voter check-in process:

- *Consume* an authorization card to prevent multiple check-ins.
- Use linear implication, $A \multimap \{B\}$.
  - $A \multimap \{B\} \approx$ "*consume* resource A to produce B."

$$\textit{voting-auth-card} \multimap \{\textit{blank-ballot}\}$$

"If I *give* an authorization card, then I *get* a blank ballot."

“May I please see your identification?”

$$\textit{voting-auth-card} \text{ “and photo ID”} \multimap \{\textit{blank-ballot}\}$$

“If I give an auth. card and a photo ID, then I get a ballot.”

Problem:
How to express a pair of resources?

"May I please see your identification?"

voting-auth-card "and photo ID" $\multimap \{$*blank-ballot*$\}$

"If I give an auth. card and a photo ID, then I get a ballot."

Problem:

How to express a pair of resources?

Solution:

Use simultaneous conjunction, $A \otimes B$.

- $A \otimes B \approx$ "both resources A and B"

“May I please see your identification?”

$$\textit{voting-auth-card} \otimes \textit{photo-ID} \multimap \{\textit{blank-ballot}\}$$

“If I give an auth. card and a photo ID, then I get a ballot.”

Problem:

How to express a pair of resources?

Solution:

Use simultaneous conjunction, $A \otimes B$.

- $A \otimes B \approx$ “both resources A and B”

“May I please see your identification?”

$$\textit{voting-auth-card} \otimes \textit{photo-ID} \multimap \{\textit{blank-ballot}\}$$

“If I give an auth. card and a photo ID, then I get a ballot.”

Problem:

How to express a pair of resources?

Solution:

Use simultaneous conjunction, $A \otimes B$.

- $A \otimes B \approx$ “both resources A and B”

Problem:

Linear implication incorrectly consumes the photo ID.
In linear logic, how do we only *show* the ID?

“May I please see your identification?”

$$\textit{voting-auth-card} \otimes \textit{photo-ID} \multimap \{\textit{blank-ballot}\}$$

“If I give an auth. card and a photo ID, then I get a ballot.”

Problem:

How to express a pair of resources?

Solution:

Use simultaneous conjunction, $A \otimes B$.

- $A \otimes B \approx$ “both resources A and B”

Problem:

Linear implication incorrectly consumes the photo ID.
In linear logic, how do we only *show* the ID?

“May I please see your identification?”

$$\textit{voting-auth-card} \otimes \,!\textit{photo-ID} \multimap \{\textit{blank-ballot}\}$$

“If I give an auth. card and show a photo ID, then I get a ballot.”

Problem:

How to express a pair of resources?

Solution:

Use simultaneous conjunction, $A \otimes B$.

- $A \otimes B \approx$ “both resources A and B”

Problem:

Linear implication incorrectly consumes the photo ID.
In linear logic, how do we only *show* the ID?

Ensuring that the Card and ID Match

$$\textit{voting-auth-card} \otimes !\textit{photo-ID} \multimap \{\textit{blank-ballot}\}$$

"If I give an auth. card and show a photo ID, then I get a ballot."

Problem:

Doesn't ensure that auth. card and photo ID match.

Ensuring that the Card and ID Match

$$\textit{voting-auth-card} \otimes !\textit{photo-ID} \multimap \{\textit{blank-ballot}\}$$

"If I give an auth. card and show a photo ID, then I get a ballot."

Problem:

Doesn't ensure that auth. card and photo ID match.

Solution:

Use universal quantification, $\forall x.A$.

- Quantified variables are not resources.

Ensuring that the Card and ID Match

$$\forall v.\, \textit{voting-auth-card}(v) \otimes !\textit{photo-ID}(v) \multimap \{\textit{blank-ballot}\}$$

"If I give an auth. card and show a matching ID, then I get a ballot

Problem:

Doesn't ensure that auth. card and photo ID match.

Solution:

Use universal quantification, $\forall x.A$.

- Quantified variables are not resources.

Ensuring that the Card and ID Match

$$\textit{voting-auth-card}(V) \otimes !\textit{photo-ID}(V) \multimap \{\textit{blank-ballot}\}$$

"If I give an auth. card and show a matching ID, then I get a ballot

Problem:

Doesn't ensure that auth. card and photo ID match.

Solution:

Use universal quantification, $\forall x.A$.

- Quantified variables are not resources.

Ensuring that the Card and ID Match

$$\textit{voting-auth-card}(V) \otimes !\textit{photo-ID}(V) \multimap \{\textit{blank-ballot}\}$$

"If I give an auth. card and show a matching ID, then I get a ballot

Problem:

Doesn't ensure that auth. card and photo ID match.

Solution:

Use universal quantification, $\forall x.A$.

- Quantified variables are not resources.

Problem:

Doesn't ensure that the auth. card and ID are mine.

Table of Contents

Celf

The Concurrent Logical Framework

Substructural Logics

$$\frac{A_1, \ldots, A_m}{B_1, \ldots, B_n}\ \textit{name}$$

- In LLF order matters [Girard '89, Cervesato et al '96]

$$\textit{name} : A_1 \otimes \cdots \otimes A_m \multimap B_1 \otimes \cdots \otimes B_n$$

- In CLF order does not matter [Cervesato et al '02]

$$\textit{name} : A_1 \otimes \cdots \otimes A_m \multimap \{B_1 \otimes \cdots \otimes B_n\}$$

Execution as Proof Search

- Proof search

$$\begin{array}{c}\textsf{send (vote } O)\\ \vdots \\ \textsf{receive (return\_code } R)\end{array}$$

corresponds to inhabitation of CLF types.

$$\textsf{send (vote } O) \multimap \{\textsf{receive (return\_code } R)\}$$

- All terms are equal modulo interleavings
- No leftovers in the multi-set allowed
- Focusing [Andreoli '93, Chaudhuri '06, Miller '05]

CLF — Types and Kinds

- LLF + concurrency monad [Harper et al '93]
- Types:

$$
\begin{aligned}
A, B &::= A \multimap B \mid \Pi x : A.\ B \mid A \,\&\, B \mid \top \mid \{S\} \mid P \\
P &::= a \mid P\ N \\
S &::= S_1 \otimes S_2 \mid 1 \mid \exists x : A.\ S \mid A
\end{aligned}
$$

- Kinds:

$$K ::= \mathsf{type} \mid \Pi x : A.\ K$$

We write $A \to B$ for $\Pi x : A.\ B$ if x does not occur in B.

CLF — Terms

Term syntax:

$$
\begin{array}{rll}
N ::= & \widehat{\lambda} x.\ N \mid \lambda x.\ N \mid \langle N_1, N_2 \rangle \mid \langle\rangle \mid \{E\} \mid & \\
& c \mid x \mid N_1 \widehat{\ } N_2 \mid N_1\ N_2 \mid \pi_1\ N \mid \pi_2\ N & \textit{Objects} \\
E ::= & \text{let } \{p\} = N \text{ in } E \mid M & \textit{Expressions} \\
M ::= & M_1 \otimes M_2 \mid 1 \mid [N, M] \mid N & \textit{Monadic objects} \\
p ::= & p_1 \otimes p_2 \mid 1 \mid [x, p] \mid x & \textit{Patterns}
\end{array}
$$

Equality: α, β, η and let-floating

$$
\begin{array}{l}
\text{let } \{p_1\} = N_1 \text{ in let } \{p_2\} = N_2 \text{ in } E \equiv \\
\text{let } \{p_2\} = N_2 \text{ in let } \{p_1\} = N_1 \text{ in } E
\end{array}
$$

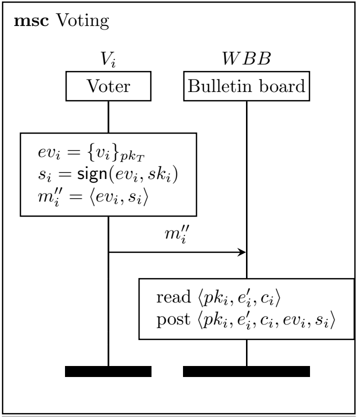
Table of Contents

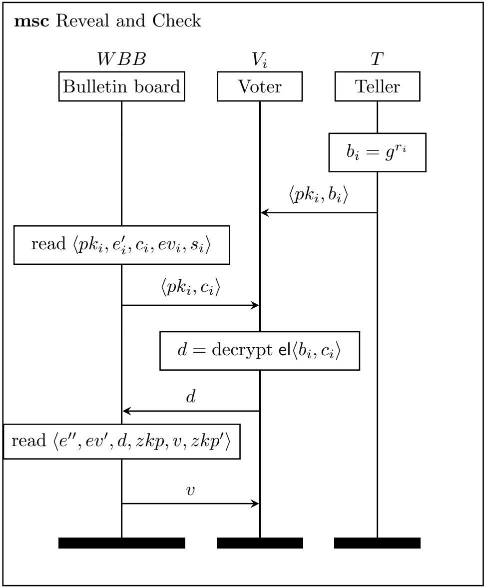
Recall the Selene Protocol

Voting in seven easy steps

1. Election Authority produces a tracker number n_i and its encryption e_i for each Voter i;
2. Mixnet shuffles the encrypted trackers e_i, resulting in a re-encryption e'_i that loses connection to n_i;
3. Teller(s) decrypt e_is, assign them to Voters V_i and generate Pedersen commitments c_i, then publish them to the Bulletin Board
4. Votes v_i are encrypted (ev_i) and signed (s_i) by Voters V_i, and published along
5. Encrypted tracking numbers and votes $\langle e'_i, ev_i \rangle$ are shuffled by the Mixnet, then published as $\langle e''_i, ev'_i \rangle$, losing link to the originals;
6. Votes ev_i are decrypted by the Tellers, and published to the Bulletin Board
7. Commitments are revealed by the Teller(s) to the Voters, who can check that their vote has been casted.

Voting and Checking





A Selene Voter in Celf

```
V : vote I PT WBB C ⊸
  { Exists r.
    net (pk I) WBB (+ (elgamal (option C) PT r)
                      (sig (elgamal (option C) PT r) I)) *
    ( Pi M1. net PT (pk I) M1 ⊸    % randomness
      Pi M2. net WBB (pk I) M2 ⊸  % trap door commitment
      Pi V. eval (dec (construct M1 M2) I) V →
      Pi V1. Pi V2. publish (+3 !V1 !V2 !(+ V (option C))) →
      { 1 }
    )
  }.
```

What can we prove?

Adequacy!

Theorem

There exists a bijection between valid traces of this protocol and (canonical) objects of type

$$\cdot \vdash N : \ldots \texttt{vote}\ V_1\ C_1 \multimap \ldots \texttt{vote}\ V_n\ C_n \multimap \ldots \multimap \{1\}$$

- Election Authority (EA), Web Bulletin Board (WBB), Mixnet (M) and Tellers (T) can be modeled similarly
- Celf allows us to experiment with such designs
- We characterize in Celf precisely the protocol that we want, not more, not less
- Execution may require complex reasoning

Coherence!

- Concept originating from Multiparty Session Types [Honda, Yoshida, Carbone 2008]
- Correspondence between linear logic propositions and session types [Carbone et al. 2015, 2016]

Coherence!

- Concept originating from Multiparty Session Types [Honda, Yoshida, Carbone 2008]
- Correspondence between linear logic propositions and session types [Carbone et al. 2015, 2016]
- Coherence ensures that the types of multiple processes are dual to each other

Coherence!

- Concept originating from Multiparty Session Types [Honda, Yoshida, Carbone 2008]
- Correspondence between linear logic propositions and session types [Carbone et al. 2015, 2016]
- Coherence ensures that the types of multiple processes are dual to each other
- i.e. no wrong execution is possible, where one process is stuck (remember: linear logic requires emptying the linear context)

Coherence!

- Concept originating from Multiparty Session Types [Honda, Yoshida, Carbone 2008]
- Correspondence between linear logic propositions and session types [Carbone et al. 2015, 2016]
- Coherence ensures that the types of multiple processes are dual to each other
- i.e. no wrong execution is possible, where one process is stuck (remember: linear logic requires emptying the linear context)
- Good first sanity check on the protocol design

Coherence!

- Concept originating from Multiparty Session Types [Honda, Yoshida, Carbone 2008]
- Correspondence between linear logic propositions and session types [Carbone et al. 2015, 2016]
- Coherence ensures that the types of multiple processes are dual to each other
- i.e. no wrong execution is possible, where one process is stuck (remember: linear logic requires emptying the linear context)
- Good first sanity check on the protocol design
- In the presence of an attacker?
  Sessions and Separability in Security Protocols [Carbone, Guttman 2013]

Demo time!

Table of Contents

Contributions

Framework perspective

- Logical frameworks support adequate encodings of complex security protocols
- Coherence check on the protocol design

Contributions

Framework perspective

- Logical frameworks support adequate encodings of complex security protocols
- Coherence check on the protocol design

Contributions

Framework perspective

- Logical frameworks support adequate encodings of complex security protocols
- Coherence check on the protocol design

Model perspective

- First coherent formalisation of selene!
- Helped to clarify what messages are exchanged when, what are the phases

What we are missing?

What we are missing?

Framework perspective

- At the moment the framework lacks **coinduction**
- Impossible to construct **indistinguishability (bisimulation)** proofs without

What we are missing?

Framework perspective

- At the moment the framework lacks **coinduction**
- Impossible to construct **indistinguishability (bisimulation)** proofs without

Model perspective

- Introducing Zero-knowledge proofs
- Express more security properties with dependent types
- Deriving **real world implementations** from the generated processes
- Deriving **models** for other tools