Automated Analysis of Non-interference Security by Refinement

Thai Son Hoang ¹  Annabelle McIver ²  Larissa Meinicke ²  Anthony Sloane ²  Enrico Susatyo ²

¹ Swiss Federal Institute of Technology Zürich (ETH Zürich), Switzerland
² Macquarie University, Sydney, Australia

21st June 2011, CryptoForma Workshop
(adapted from slides by Annabelle McIver)
Secure Refinement

- Specialisation of classical refinement;
- Preserves non-interference security properties;
- It is compositional;
- It supports hierarchical program development;
- Its semantics provides a link between “source code” and the “mathematics underlying secrecy”.
Secure Refinement-oriented Approach
A short history (1/2)

- Traditional refinement reduces non-determinism, preserving all “relevant properties”.
  \[ P \cap Q \subseteq P \]

- Traditional formal approaches to security model a “secret” as a non-deterministic choice over its “type”.

- Refinement paradox:
  \[
  h :\in \{0, 1\} \quad \not\subseteq \quad h := 0
  \]
  \[
  h :\in \{0, 1\} \quad \not\subseteq_{\text{secure}} \quad h := 0
  \]

- Traditional refinement is defined relative to a flat state space.

- Secure refinement uses a structured state space.
A secret is an **undisclosed** choice over a set of possibilities.

A non-deterministic choice is a **disclosed** choice, with the selection made as a program is developed.

The two choices should be distinguished in the semantics.

- Undisclosed choice **cannot** (accidentally) be “refined away”,
- so that refinements preserve secrecy.
1. During program execution, after each “atomic step”:
   - can “look” at the visible variables
   - cannot “look” at the hidden variables

2. Can observe any branching.

(1) and (2) imply compositionality of refinement.

A qualitative approach: “run the program only once”.
Hidden/Visibles in the Programming Language

- $v$ (of type $\mathcal{V}$) is visible, $h$ (of type $\mathcal{H}$) is hidden.
- $H$ (of type $\mathcal{P}(\mathcal{H})$) – the shadow – the set of possible values of $h$.
- Assume: $v, h \in \{0, 1\}$, initially $H$ is $\{0, 1\}$.

Program $(v', h', H')$

Set hidden

$h := 0$

$\{(v, 0, \{0\})\}$

$h \in \{0, 1\}$

$\{(v, 0, \{0, 1\}), (v, 1, \{0, 1\})\}$

Set visible

$v := 0$

$\{(0, h, \{0, 1\})\}$

$v \in \{0, 1\}$

$\{(0, h, \{0, 1\}), (1, h, \{0, 1\})\}$

Swap hidden

$h \in \{0, 1\}; h := 1 - h$

$\{(v, 0, \{0, 1\}), (v, 1, \{0, 1\})\}$
In *secure refinement-oriented* framework:

- we do not say that “a program is secure”,
- we write a *specification* which “obviously” captures our requirements (both functional and security),
- specification summarises the *intentions* of the designer: inefficient or unimplementable “programs”.
- we use *refinement* to add detail.
- Result: avoid building insecurities into the system.
Event-B: modelling discrete transition systems using refinement.

Event-B is supported by the Rodin Platform.

A specialised refinement is implemented for the Rodin platform.

An extra variable $H$ (the “Shadow”) is generated to keep track of the possible values of hidden variables $h$.

Extra refinement relations for shadow refinement.

Rodin generates and discharges many of the obligations related to shadow refinement.

Interactively prove the remaining obligations within Rodin.
Difficulty: it was awkward to generate and supply the invariants for the shadow $H$.

Solution: Implemented a “front-end” for inputting program directly, using Rodin as a “back-end” for verification.

The shadow invariants are generated in Rodin.
HID $E : X$

result: skip;

[= 

VIS $v : X$
HID $h : X$
FUN $\oplus : X \times X \rightarrow X$

result: $v = h \oplus E$

variables: $E, \text{fresult}, H1$

result
when
$\text{fresult} = F$
then
$\text{fresult} := T$
end

invariants:
$E \in H1$
$\text{fresult} = F \Rightarrow H1 = X$
$\text{fresult} = T \Rightarrow (\forall vb \cdot vb \in H1 \Rightarrow vb \in X)$
Can We Automate These Proofs? (3/4)

HID E : X
result: skip;

VIS v : X
HID h : X
FUN ⊕ : X x X -> X
result: v = h ⊕ E

variables: E, fresult, H1

result
when fresult = F
then fresult := T
end

invariants:
E ∈ H1
fresult = F ⇒ H1 = X
fresult = T ⇒ (∀vb. vb ∈ H1 ⇒ vb ∈ X)
Can We Automate These Proofs? (3/4)

variables:  $E$, $fresult$, $H1$

result: skip;

$[=\$

VIS $v : X$
HID $h : X$
FUN $\oplus : X \times X \to X$

result: $v = h \oplus E$

invariants:

$E \in H1$

$fresult = F \Rightarrow H1 = X$

$fresult = T \Rightarrow (\forall vb. vb \in H1 \Rightarrow vb \in X)$
variables: $E, v, h, \text{fresult}, H2$

result

when

\[ \text{fresult} = F \]

then

\[ \text{fresult} := T \]

\[ v := h \oplus E \]

\[ H2 := \{ vE \mapsto vh \in H2 \mid h \oplus E = vh \oplus vE \} \]

end

invariants:

\[ E \mapsto h \in H2 \]
\[ \text{fresult} = F \Rightarrow H2 = X \times X \]
\[ \text{fresult} = T \Rightarrow (\forall vE \mapsto vh \in H2 \mid v = vh \oplus vE) \]
\[ \forall vE, vE' \in H1 \Rightarrow (\exists vh, vE \oplus vh \in H2) \]
Can We Automate These Proofs? (4/4)

**variables:** \( E, \nu, h, \text{fresult}, H2 \)

\[
\begin{align*}
\text{result: } & \text{skip; } \\
\text{when } & f\text{result} = F \\
\text{then } & f\text{result} := T \\
\nu & := h \oplus E \\
H2 & := \{ vE \mapsto vh \in H2 \mid h \oplus E = vh \oplus vE \}
\end{align*}
\]

\textbf{invariants:}

\[
\begin{align*}
E & \mapsto h \in H2 \\
f\text{result} = F & \Rightarrow H2 = X \times X \\
f\text{result} = T & \Rightarrow (\forall vE \mapsto vh \in H2 \cdot \nu = vh \oplus vE) \\
\forall vE \cdot vE \in H1 & \Rightarrow (\exists vh \cdot vE \mapsto vh \in H2)
\end{align*}
\]
variables: \( E, v, h, fresult, H2 \)

result

when

\( fresult = F \)

then

\( fresult := T \)
\( v := h \oplus E \)
\( H2 := \{ vE \leftrightarrow vh \in H2 \mid h \oplus E = vh \oplus vE \} \)

end

invariants:

\( E \leftrightarrow h \in H2 \)
\( fresult = F \Rightarrow H2 = X \times X \)
\( fresult = T \Rightarrow ( \forall vE \leftrightarrow vh \in H2 \cdot v = vh \oplus vE ) \)
\( \forall vE \cdot vE \in H1 \Rightarrow ( \exists vh \cdot vE \leftrightarrow vh \in H2 ) \)
variables: $E, v, h, \text{fresult}, H2$

result: skip;

\begin{align*}
\text{when} & \quad \text{fresult} = F \\
\text{then} & \quad \text{fresult} := T \\
& \quad v := h \oplus E \\
& \quad H2 := \{vE \leftrightarrow vh \in H2 \mid h \oplus E = vh \oplus vE\}
\end{align*}

end

invariants:
\begin{align*}
E & \leftrightarrow h \in H2 \\
\text{fresult} = F & \Rightarrow H2 = X \times X \\
\text{fresult} = T & \Rightarrow (\forall vE \leftrightarrow vh \in H2 \cdot v = vh \oplus vE) \\
\forall vE \cdot vE & \in H1 \Rightarrow (\exists vh \cdot vE \leftrightarrow vh \in H2)
\end{align*}
Conclusions and Future Work

- We shown how to automate Shadow refinement proofs using Event-B/Rodin.
- The proofs are valid for a restricted sub-sets of language of probabilistic model.
- Future work:
  - Better integration tool support.
  - Applications to other protocols.