Verifying a Hotel Key Card System

Tobias Nipkow, ICTAC 2006
Presentation by: Hossein Hojjat

EPFL

April 30, 2009

Outline

Hotel Key Card

- Decentralized system
- Two key numbers in a card
  - key_1: old key of the previous occupant
  - key_2: new key of the current occupant
- One key number in a lock
  - $key_{\mathrm{L}} = key_2$: Open
  - $key_{\mathrm{L}} = key_1$: Open & Recode $key_{\mathrm{L}} := key_2$

Hotel Key Card

k_1

Hotel Key Card

$$k_1 \xrightarrow{(k_1,k_2)} k_2$$

Hotel Key Card

$$k_1 \xrightarrow{(k_1,k_2)} k_2 \xrightarrow{(k_1,k_2)} k_2$$

Hotel Key Card

$$k_1 \xrightarrow{(k_1,k_2)} k_2 \xrightarrow{(k_1,k_2)} k_2 \xrightarrow{(k_2,k_3)} k_3 \cdots$$

Correctness

- Is the system correct?
- Safety: Only the owner of a room can be in a room
- Liveness?
- Verify the correctness of the system using Alloy and Isabelle/HOL
  - Alloy implementation is taken from "Software Abstractions: Logic, Language, and Analysis", Daniel Jackson

Outline

Objects

```
sig Key, Time {}
sig Card { fst, snd: Key }
sig Room { key: Key one → Time}
one sig Desk {
   issued: Key → Time,
   prev: (Room → lone Key) → Time}
sig Guest {
 cards: Card → Time}
```

```
pred init [t: Time] {
   Desk.prev.t = key.t
   Desk.issued.t = Room.key.t and no cards.t }
```

Checkin

```
pred checkin [t,t': Time, r: Room, g: Guest] {
 some c: Card {
    c.fst = r.(Desk.prev.t)
    c.snd not in Desk.issued.t
    cards.t' = cards.t + g → c
    Desk.issued.t' = Desk.issued.t + c.snd
    Desk.prev.t' = Desk.prev.t ++ r → c.snd
    }
 key.t = key.t'
 }
```

Enter

```
pred enter [t,t': Time, r: Room, g: Guest] {
 some c: g.cards.t |
    let k = r.key.t {
    c.snd = k and key.t' = key.t
    or c.fst = k and key.t' = key.t ++ r → c.snd
    }
 issued.t = issued.t' and prev.t = prev.t'
 cards.t = cards.t'
 }
```

Demo (Allay)

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)
Check-out	G_1	

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)
Check-out	G_1	
Check-in	G_2	(k_2, k_3)

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)
Check-out	G_1	
Check-in	G_2	(k_2, k_3)
Check-out	G_2	

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)
Check-out	G_1	
Check-in	G_2	(k_2, k_3)
Check-out	G_2	
Check-in	G_1	(k_3, k_4)

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)
Check-out	G_1	
Check-in	G_2	(k_2, k_3)
Check-out	G_2	
Check-in	G_1	(k_3, k_4)
Enter-room	G_1	(k_1, k_2)

Guest-in-the-middle attack

Check-in	G_1	(k_1, k_2)
Check-out	G_1	
Check-in	G_2	(k_2, k_3)
Check-out	G_2	
Check-in	G_1	(k_3, k_4)
Enter-room	G_1	(k_1, k_2)
Enter-room	G_2	(k_2, k_3)

General Case

Alloy solution

- Assume everybody returns their old cards upon check-in
- cards.t' $=$ cards.t $+$ g $\rightarrow$ c
- cards.t' $=$ cards.t $++$ g $\rightarrow$ c

Theorem proving

- Alloy conjecture: No attack for 4 keys and cards, 7 time instants, two guests and one room
- Prove the conjecture in Isabelle/HOL

Outline

Record state

(* reception *)

owns	::	$room \Rightarrow guest$
currk	::	$room \Rightarrow key$
issued	::	$key\ set$

(* guests *)

cards	::	$guest \Rightarrow card\ set$

(* rooms *)

roomk	::	$room \Rightarrow key$
isin	::	$room \Rightarrow guest\ set$

Initialization

$$\begin{aligned}
\llparenthesis\ & owns = arbitrary, \\
& currk = initk, \\
& issued = range\ initk, \\
& cards = (\lambda g.\emptyset), \\
& roomk = initk, \\
& isin = (\lambda r.\emptyset), \\
\rrparenthesis & \in R
\end{aligned}$$

Check-in

$$
\begin{array}{l}
s \in R \text{ and } k \notin \mathit{issued}\ s \text{ then} \\
\llparenthesis\ \ \mathit{owns} := (\mathit{owns}\ s)(r := g), \\
\quad \mathit{cards} := (\mathit{cards}\ s)(g := \mathit{cards}\ s\ g \cup \{(\mathit{currk}\ s\ r, k)\}), \\
\quad \mathit{currk} := (\mathit{currk}\ s)(r := k), \\
\quad \mathit{issued} := \mathit{issued}\ s \cup \{\mathrm{k}\} \\
\rrparenthesis \in R
\end{array}
$$

Enter room

$$
\begin{array}{l}
s \in R \text{ and } (k, k') \in cards\ s\ g \text{ and } roomk\ s\ r \in \{k, k'\} \text{ then} \\
\llparenthesis\ \ isin := (isin\ s)(r := isin\ s\ r \cup \{g\}), \\
\quad roomk := (roomk\ s)(r := k') \\
\rrparenthesis \in R
\end{array}
$$

Safety formalized

- Add state component *safe* :: *room* ⇒ *bool*
- Initially *safe* is *True* everywhere
- Check-in for room *r* sets *safe r* to *False*
- Enter for room *r* sets *safe r* to *True*
  if the owner entered an empty room
  with card $(-, k')$ such that k' is *currk r* (at reception)
- Proof: If a room is *safe*, only its owner can be in it

Demo (Isabelle/HOL)

Two approaches

Alloy	Isabelle/HOL
Software specs	General purpose
Set theory	Higher-Order Logic
Search for finite counter examples	Interactive & automatic proof