State and Protocols: The Envelope Example

rigorous design for protocols using state

Daniel J. Dougherty and Joshua D. Guttman

Worcester Polytechnic Institute

Thanks to: National Science Foundation
(Grant CNS-0952287).

Mar 2013

Goal of this Talk

Illustrate a:

Logical approach to
protocol refinement
for protocols using state via
Hardware Security Modules

(specifically Trusted Platform Modules)

using M. Ryan's "envelope protocol"

Goal of this Talk

Illustrate a:

Diagrammatic approach to
protocol refinement
for protocols using state via
Hardware Security Modules

(specifically Trusted Platform Modules)

using M. Ryan's "envelope protocol"

Goal of this Talk

Illustrate a:

Diagrammatic approach to
protocol refinement
for protocols using state via
Hardware Security Modules

(specifically Trusted Platform Modules)

using M. Ryan's "envelope protocol"

Trusted Platform Modules

- Small cheap chip on motherboard of many PCs
- Offers:
  - Cryptographic primitives
  - Some protected storage
  - Platform configuration registers
    that record certain event sequences
- State is a reliable record of those events
- Supports attestation

Signed assertions about TPM state,
reflecting system history

The Envelope Protocol

Alice deposits an encrypted secret $\{\!|\, v \,|\!\}_K$ s.t.

- Bob can obtain v
- Or Bob can certify v was never obtained, and never will be obtained
- But not both

Bob may be adversarial, misbehaving to try to get both v and refuse($\{\!|\, v \,|\!\}_K$)

TPM used to achieve protocol goal

The Envelope Protocol

Alice deposits an encrypted secret $\{\!|\, v \,|\!\}_K$ s.t.

- Bob can obtain v
- Or Bob can certify v was never obtained, and never will be obtained
- But not both

Bob may be adversarial, misbehaving to try to get both v and refuse($\{\!|\, v \,|\!\}_K$)

TPM used to achieve protocol goal

The Envelope Protocol

Alice deposits an encrypted secret $\{\!|\, v \,|\!\}_K$ s.t.

- Bob can obtain v
- Or Bob can certify v was never obtained, and never will be obtained
- But not both

Bob may be adversarial, misbehaving to try to get both v and refuse($\{\!|\, v \,|\!\}_K$)

TPM used to achieve protocol goal

The Envelope Protocol

Alice deposits an encrypted secret $\{\!|\, v \,|\!\}_K$ s.t.

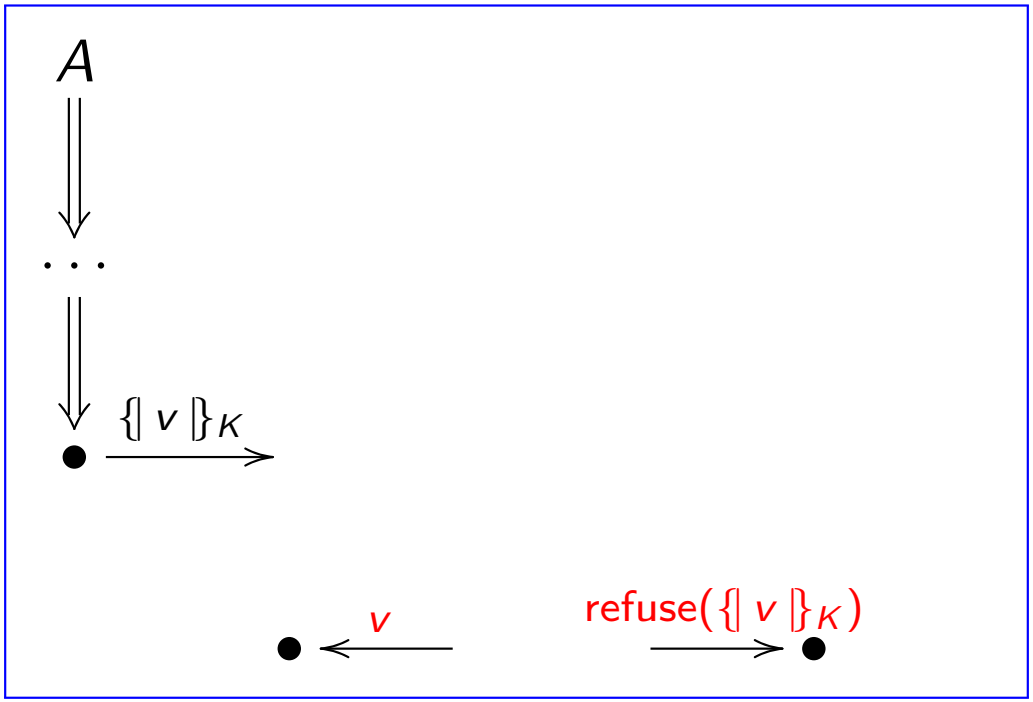
- Bob can obtain v
- Or Bob can certify v was never obtained, and never will be obtained
- But not both

Bob may be adversarial, misbehaving to try to get both v and refuse($\{\!|\, v \,|\!\}_K$)

TPM used to achieve protocol goal

Envelope Protocol: Security Goal

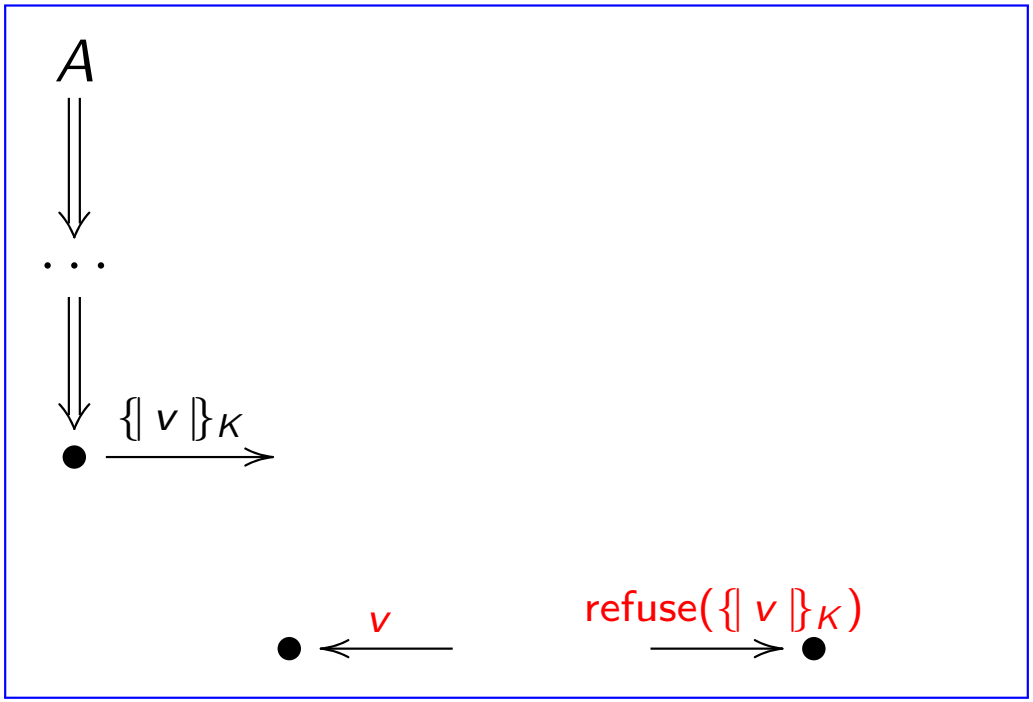
This diagram never occurs within any execution:



where v is fresh and unguessable

Envelope Protocol: Security Goal

This diagram never occurs within any execution:



where v is fresh and unguessable

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register

   Suppose: Just one TPM with just one PCR
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR
   regardless of this choice

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register

   Suppose: Just one TPM with just one PCR
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR regardless of this choice

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register

   Suppose: Just one TPM with just one PCR
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR regardless of this choice

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register

   Suppose: Just one TPM with just one PCR
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR regardless of this choice

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register

   Suppose: Just one TPM with just one PCR
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR
   regardless of this choice

Analyzing the Envelope Protocol

Delaune-Kremer-Ryan-Steel: verify it via ProVerif

Under some restrictions:

- Bounded number of reboots
- Anchored at initial, post-boot state
- Monolithic

Our contributions:

1. Lift the restrictions above

   using explicit model of distributed system behavior
2. Proof method tailored to protocol refinement steps
3. Ensure "disjoint encryption" properties

Analyzing the Envelope Protocol

Delaune-Kremer-Ryan-Steel: verify it via ProVerif
Under some restrictions:

- Bounded number of reboots
- Anchored at initial, post-boot state
- Monolithic

Our contributions:

1. Lift the restrictions above
   using explicit model of distributed system behavior
2. Proof method tailored to protocol refinement steps
3. Ensure "disjoint encryption" properties

Analyzing the Envelope Protocol

Delaune-Kremer-Ryan-Steel: verify it via ProVerif
Under some restrictions:

- Bounded number of reboots
- Anchored at initial, post-boot state
- Monolithic

Our contributions:

1. Lift the restrictions above

   using explicit model of distributed system behavior
2. Proof method tailored to protocol refinement steps
3. Ensure "disjoint encryption" properties

Strands with state synchronization

A strand is a linear sequence of

1. transmission nodes $\bullet \xrightarrow{t}$
2. reception nodes $\bullet \xleftarrow{t}$
3. state synchronization nodes $\circ\ \phi$

representing a single local session $\bullet \Rightarrow \circ \Rightarrow \bullet$

We view transmissions as positive
receptions as negative
state synchronizations as neutral

Strands with state synchronization

A strand is a linear sequence of

1. transmission nodes $\bullet \xrightarrow{t}$
2. reception nodes $\bullet \xleftarrow{t}$
3. state synchronization nodes $\circ\ \phi$

representing a single local session $\bullet \Rightarrow \circ \Rightarrow \bullet$

We view transmissions as positive
receptions as negative
state synchronizations as neutral

Bundles, executions

Definition

A *bundle* $\mathcal{B}$ is a finite directed acyclic graph of nodes

- closed backward on each strand $\bullet \Rightarrow \circ \Rightarrow \bullet$
- every reception has a unique matching transmission $\bullet \xrightarrow{t} \bullet$

Definition

An *execution* is a pair $\mathcal{B}, \preceq$ where $\mathcal{B}$ is a bundle and $\preceq$ is a partial order extending $(\Rightarrow \cup \rightarrow)^*$ such that

- $\preceq$ linearly orders the neutral nodes
- certain state evolution rules are satisfied
- $n_1 \prec n_2$ implies $\mathrm{pcr\_after}(n_1) = \mathrm{pcr\_before}(n_2)$ or

$$\exists m \,.\, \mathrm{Neutral}(m) \wedge n_1 \prec m \prec n_2$$

Bundles, executions

Definition

A *bundle* $\mathcal{B}$ is a finite directed acyclic graph of nodes

- closed backward on each strand $\bullet \Rightarrow \circ \Rightarrow \bullet$
- every reception has a unique matching transmission $\bullet \xrightarrow{t} \bullet$

Definition

An *execution* is a pair $\mathcal{B}, \preceq$ where $\mathcal{B}$ is a bundle and $\preceq$ is a partial order extending $(\Rightarrow \cup \rightarrow)^*$ such that

- $\preceq$ linearly orders the neutral nodes
- certain state evolution rules are satisfied
- $n_1 \prec n_2$ implies $\mathrm{pcr\_after}(n_1) = \mathrm{pcr\_before}(n_2)$ or

$$\exists m \,.\, \mathrm{Neutral}(m) \wedge n_1 \prec m \prec n_2$$

Bundles, executions

Definition

A *bundle* $\mathcal{B}$ is a finite directed acyclic graph of nodes

- closed backward on each strand $\bullet \Rightarrow \circ \Rightarrow \bullet$
- every reception has a unique matching transmission $\bullet \xrightarrow{t} \bullet$

Definition

An *execution* is a pair $\mathcal{B}, \preceq$ where $\mathcal{B}$ is a bundle and $\preceq$ is a partial order extending $(\Rightarrow \cup \rightarrow)^*$ such that

- $\preceq$ linearly orders the neutral nodes
- certain state evolution rules are satisfied
- $n_1 \prec n_2$ implies $\mathrm{pcr\_after}(n_1) = \mathrm{pcr\_before}(n_2)$ or

$$\exists m \,.\, \mathrm{Neutral}(m) \wedge n_1 \prec m \prec n_2$$

Bundles, executions

Definition

A *bundle* $\mathcal{B}$ is a finite directed acyclic graph of nodes

- closed backward on each strand $\bullet \Rightarrow \circ \Rightarrow \bullet$
- every reception has a unique matching transmission $\bullet \xrightarrow{t} \bullet$

Definition

An *execution* is a pair $\mathcal{B}, \preceq$ where $\mathcal{B}$ is a bundle and $\preceq$ is a partial order extending $(\Rightarrow \cup \rightarrow)^*$ such that

- $\preceq$ linearly orders the neutral nodes
- certain state evolution rules are satisfied
- $n_1 \prec n_2$ implies $\mathrm{pcr\_after}(n_1) = \mathrm{pcr\_before}(n_2)$ or

$$\exists m \,.\, \mathrm{Neutral}(m) \wedge n_1 \prec m \prec n_2$$

Bundles, executions

Definition

A *bundle* $\mathcal{B}$ is a finite directed acyclic graph of nodes

- closed backward on each strand $\bullet \Rightarrow \circ \Rightarrow \bullet$
- every reception has a unique matching transmission $\bullet \xrightarrow{t} \bullet$

Definition

An *execution* is a pair $\mathcal{B}, \preceq$ where $\mathcal{B}$ is a bundle and $\preceq$ is a partial order extending $(\Rightarrow \cup \rightarrow)^*$ such that

- $\preceq$ linearly orders the neutral nodes
- certain state evolution rules are satisfied
- $n_1 \prec n_2$ implies $\mathrm{pcr\_after}(n_1) = \mathrm{pcr\_before}(n_2)$ or

$$\exists m \,.\, \mathrm{Neutral}(m) \wedge n_1 \prec m \prec n_2$$

I. State Evolution Equations

TPM Formalization

- Boot(n) implies

$$\text{pcr\_after}(n) = -1$$

- Extend(n, x) implies

$$\text{pcr\_after}(n) = \text{Hash}(\text{pcr\_before}(n), x)$$

- Boot(n) or Extend(n, x) or

$$\text{pcr\_after}(n) = \text{pcr\_before}(n)$$

Will write pcr(n) when

$$\text{pcr\_after}(n) = \text{pcr\_before}(n)$$

I. State Evolution Equations

TPM Formalization

- Boot(n) implies

$$\text{pcr\_after}(n) = -1$$

- Extend(n, x) implies

$$\text{pcr\_after}(n) = \text{Hash}(\text{pcr\_before}(n), x)$$

- Boot(n) or Extend(n, x) or

$$\text{pcr\_after}(n) = \text{pcr\_before}(n)$$

Will write pcr(n) when

$$\text{pcr\_after}(n) = \text{pcr\_before}(n)$$

II. Prefix/Boot

TPM Formalization

Definition

Prefix(x, y) iff, recursively,

- $x = y$ or
- $\exists z, w\ .\quad y = \mathsf{Hash}(z, w)$ and $\mathsf{Prefix}(x, z)$

Lemma

$n_0 \preceq n_2$ *implies*

either $\mathsf{Prefix}(\mathsf{pcr\_before}(n_0), \mathsf{pcr\_after}(n_2))$

or $\exists n_1\ .\ n_0 \preceq n_1 \preceq n_2$ *and* $\mathsf{Boot}(n_1)$

II. Prefix/Boot

TPM Formalization

Definition

Prefix(x, y) iff, recursively,

- $x = y$ or
- $\exists z, w\ .\quad y = \text{Hash}(z, w)$ and Prefix(x, z)

Lemma

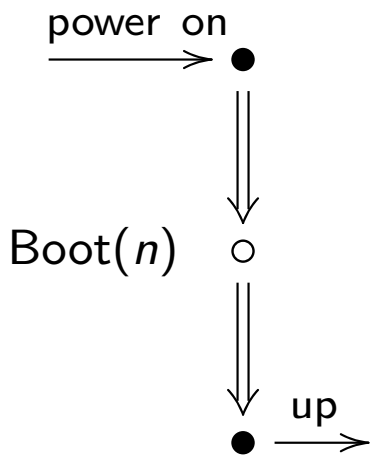
$n_0 \preceq n_2$ *implies*

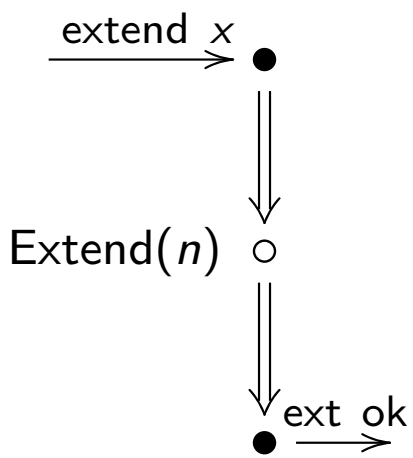
either Prefix(pcr_before(n_0), pcr_after(n_2))

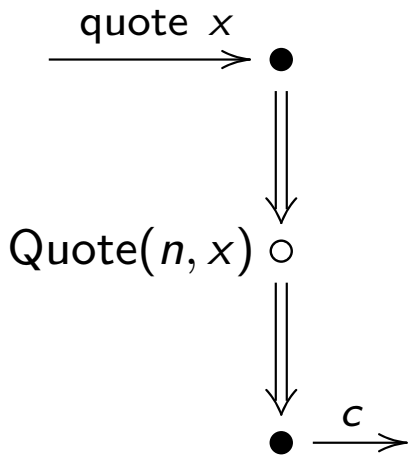
or $\exists n_1\ .\ n_0 \preceq n_1 \preceq n_2$ *and* Boot(n_1)

IIIa. Request/reply roles

TPM Formalization



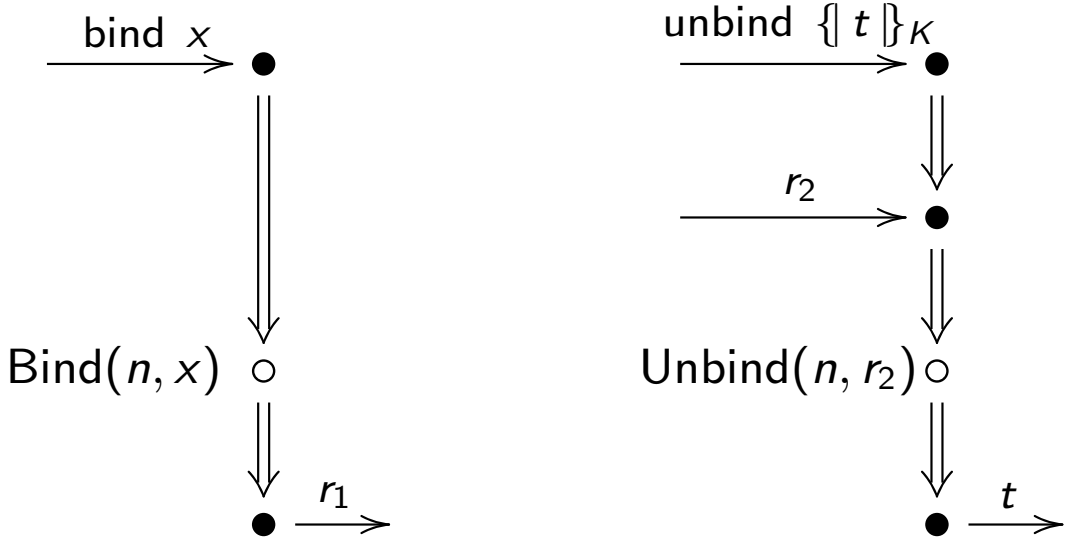




$$c = [\![\, \text{QUOTE}\ \ \text{pcr}(n),\ x \,]\!]_{\text{aik}}$$

IIIb. Request/reply roles

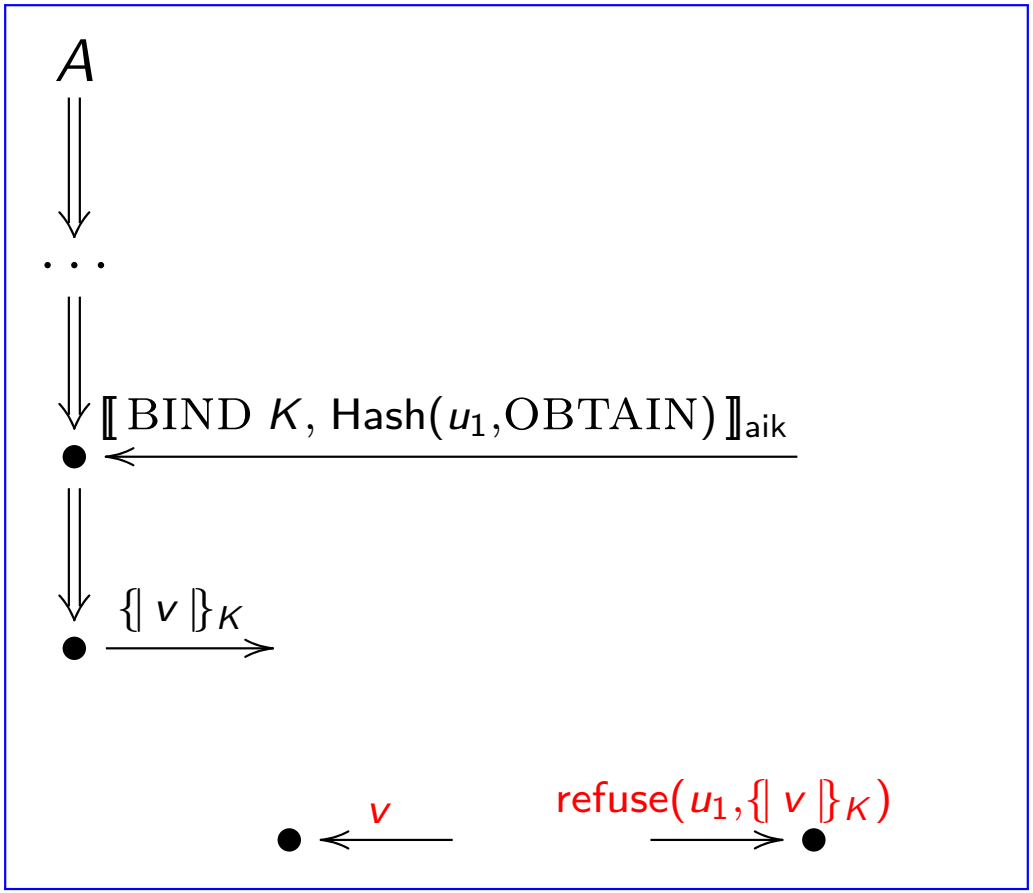
TPM Formalization



$$
\begin{aligned}
r_1 &= [\![\, \text{BIND}\ K,\ x \,]\!]_{\text{aik}} \\
r_2 &= [\![\, \text{BIND}\ K,\ \text{pcr}(n) \,]\!]_{\text{aik}} \\
K &\in \text{fresh values} \\
K^{-1} &\in \text{uncompromised, in bind role}
\end{aligned}
$$

Envelope Protocol: Refined Security Goal

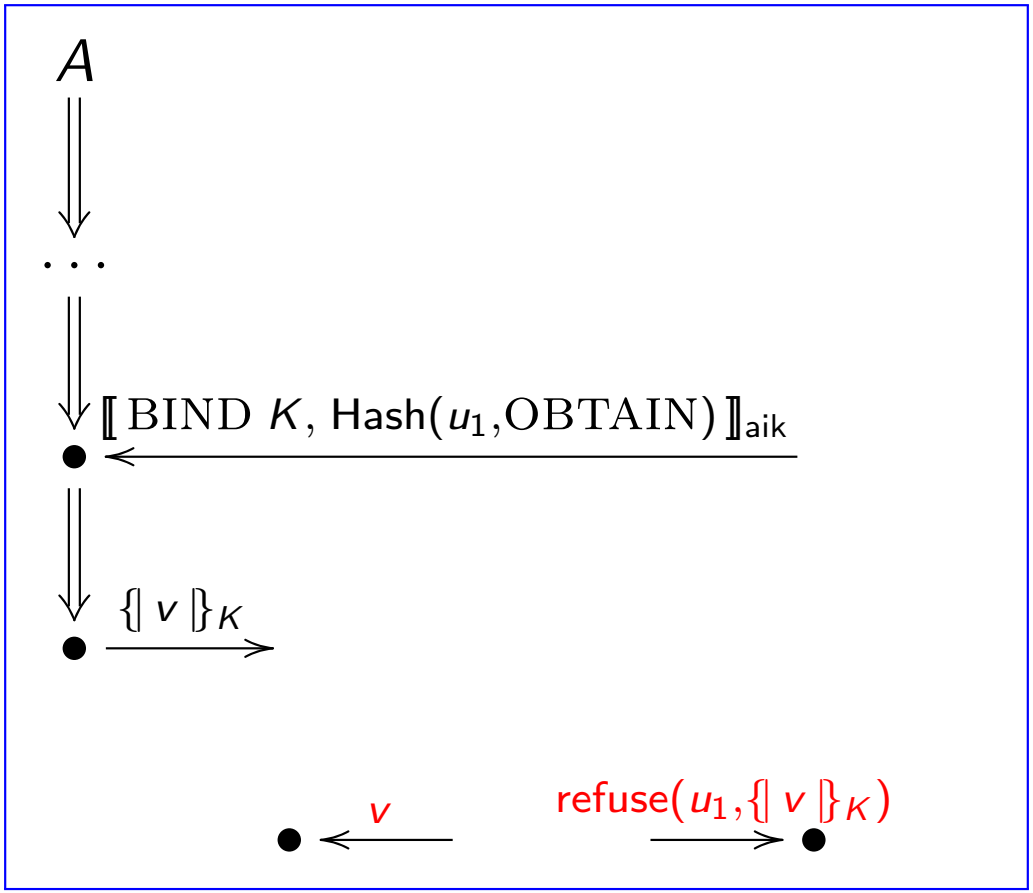
This diagram never occurs within any execution:



$$\text{refuse}(u_1, \{\!|\, v \,|\!\}_K) = [\![\, \text{QUOTE Hash}(u_1, \text{REFUSE}),\ \{\!|\, v \,|\!\}_K \,]\!]_{\text{aik}}$$

Envelope Protocol: Refined Security Goal

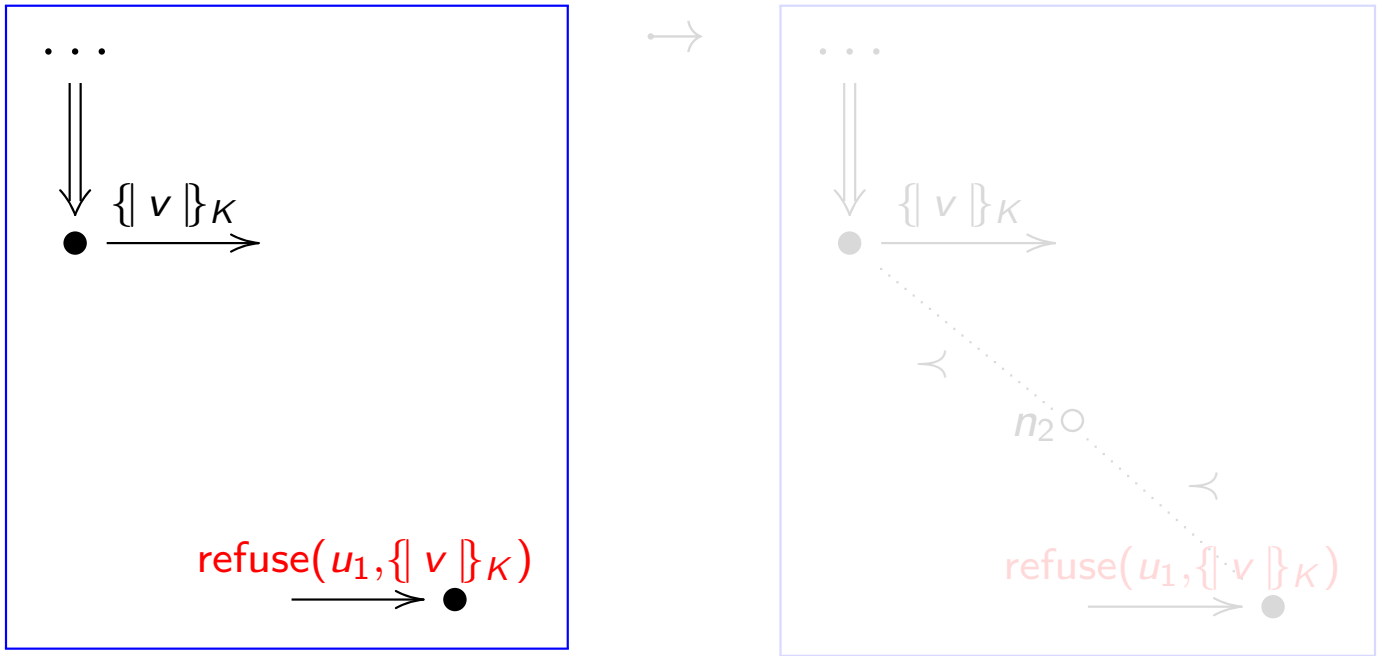
This diagram never occurs within any execution:



$$\mathsf{refuse}(u_1, \{\!|\, v \,|\!\}_K) = [\![\, \mathrm{QUOTE}\ \mathsf{Hash}(u_1, \mathrm{REFUSE}),\ \{\!|\, v \,|\!\}_K \,]\!]_{\mathsf{aik}}$$

Refusal Specification

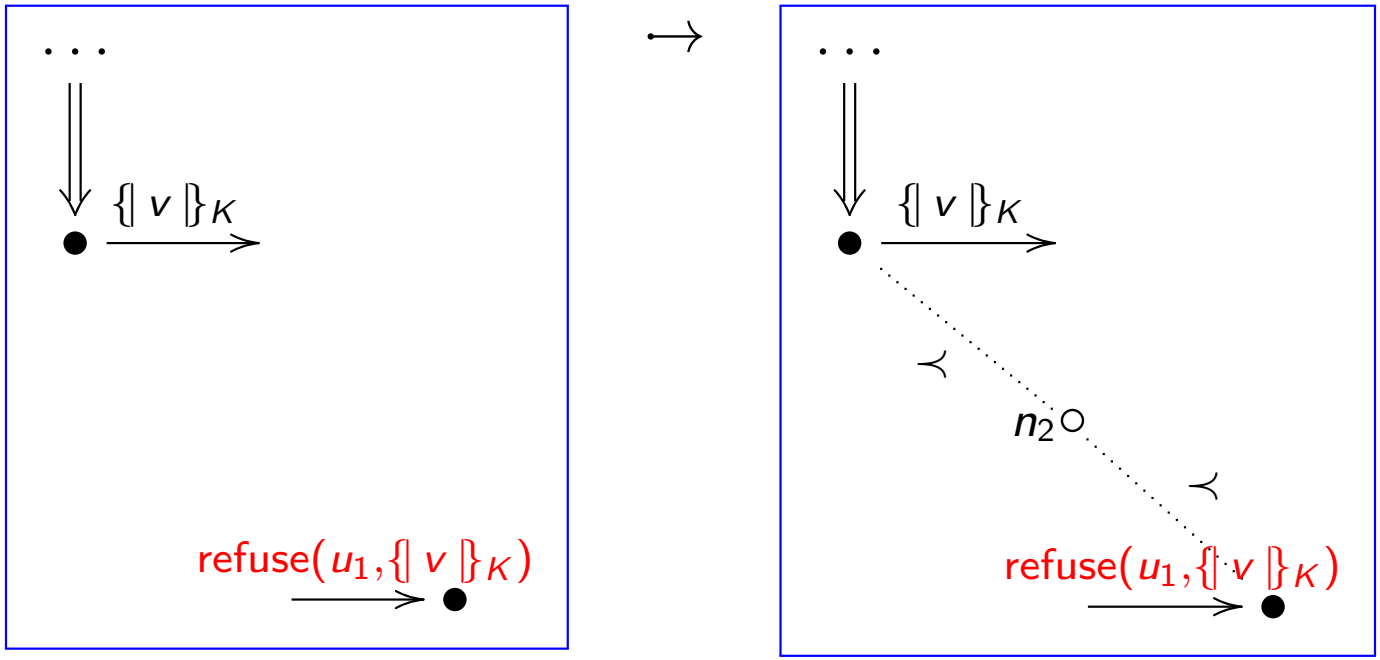
Must traverse Quote-able state



Quote$(n_2, \{\!|\, v \,|\!\}_K)$ $\quad$ pcr$(n_2) = \text{Hash}(u_1, \text{REFUSE})$

Refusal Specification

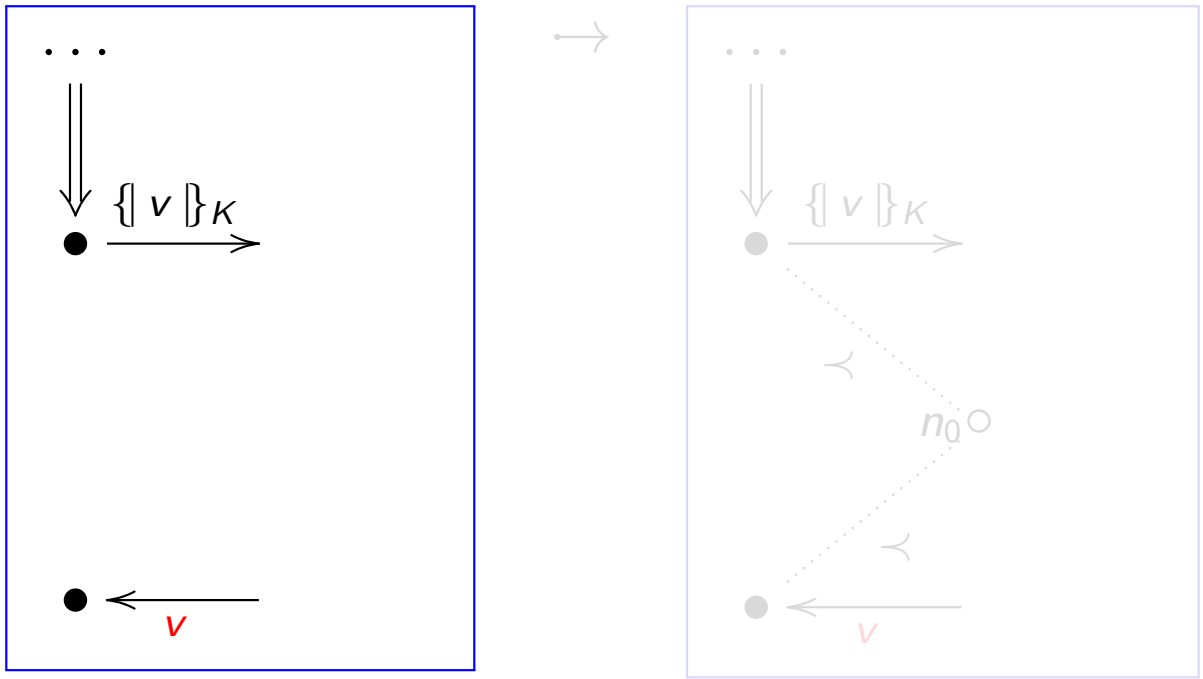
Must traverse Quote-able state



$\mathsf{Quote}(n_2, \{\!|\, v \,|\!\}_K)$ $\quad$ $\mathsf{pcr}(n_2) = \mathsf{Hash}(u_1, \mathrm{REFUSE})$

Disclosure Specification

Must traverse Unbindable state

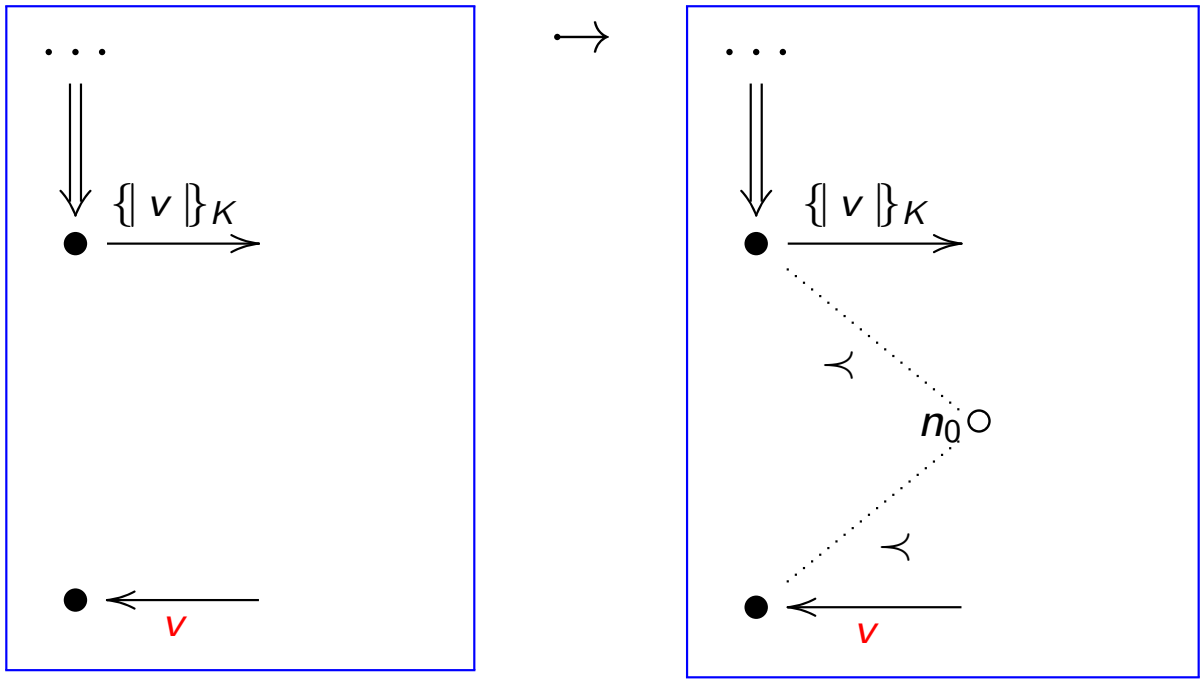


$$\text{Unbind}(n_0, [\![\ \text{BIND}\ K,\ \text{Hash}(u_1, \text{OBTAIN})\ ]\!]_{\text{aik}})$$

$$\text{so}\ \ \text{pcr}(n_0) = \text{Hash}(u_1, \text{OBTAIN})$$

Disclosure Specification

Must traverse Unbindable state

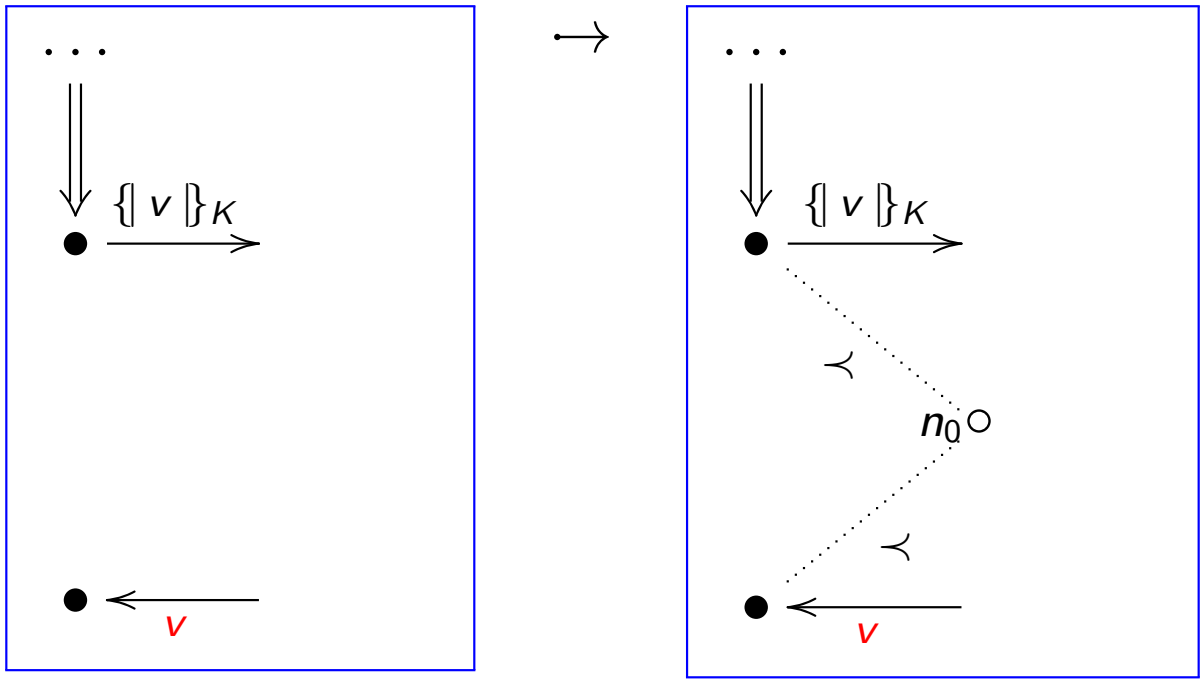


$$\text{Unbind}(n_0, [\![\ \text{BIND}\ K,\ \text{Hash}(u_1, \text{OBTAIN})\ ]\!]_{\text{aik}})$$

so $\text{pcr}(n_0) = \text{Hash}(u_1, \text{OBTAIN})$

Disclosure Specification

Must traverse Unbindable state



$$\text{Unbind}(n_0, [\![\ \text{BIND}\ K,\ \text{Hash}(u_1, \text{OBTAIN})\ ]\!]_{\text{aik}})$$

$$\text{so}\ \ \text{pcr}(n_0) = \text{Hash}(u_1, \text{OBTAIN})$$

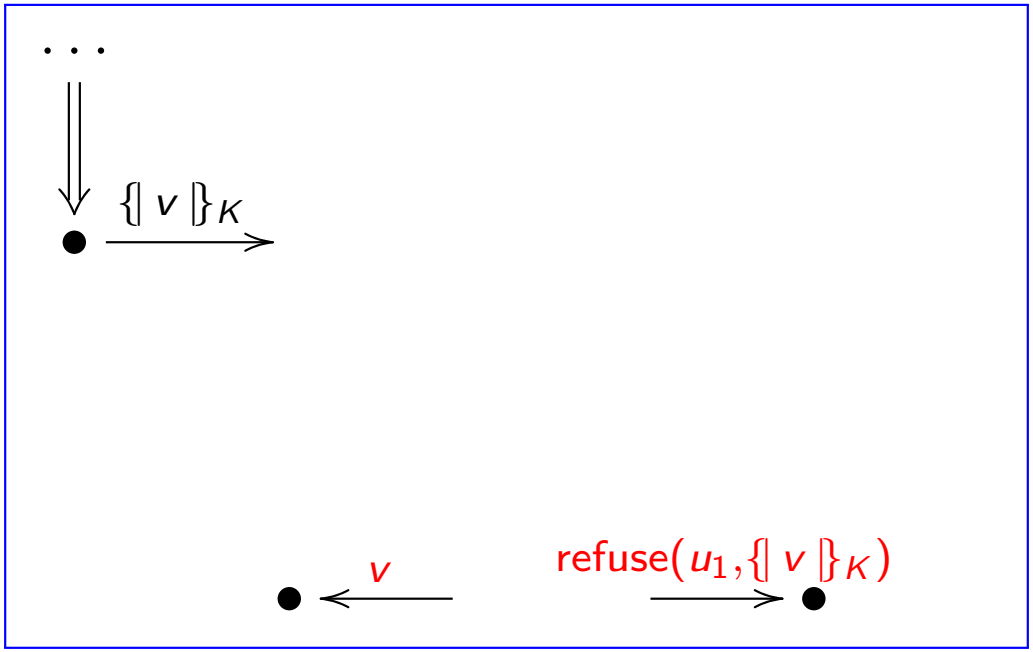
Implementation Idea

Using a TPM

1. Use one TPM platform configuration register
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR regardless of this choice

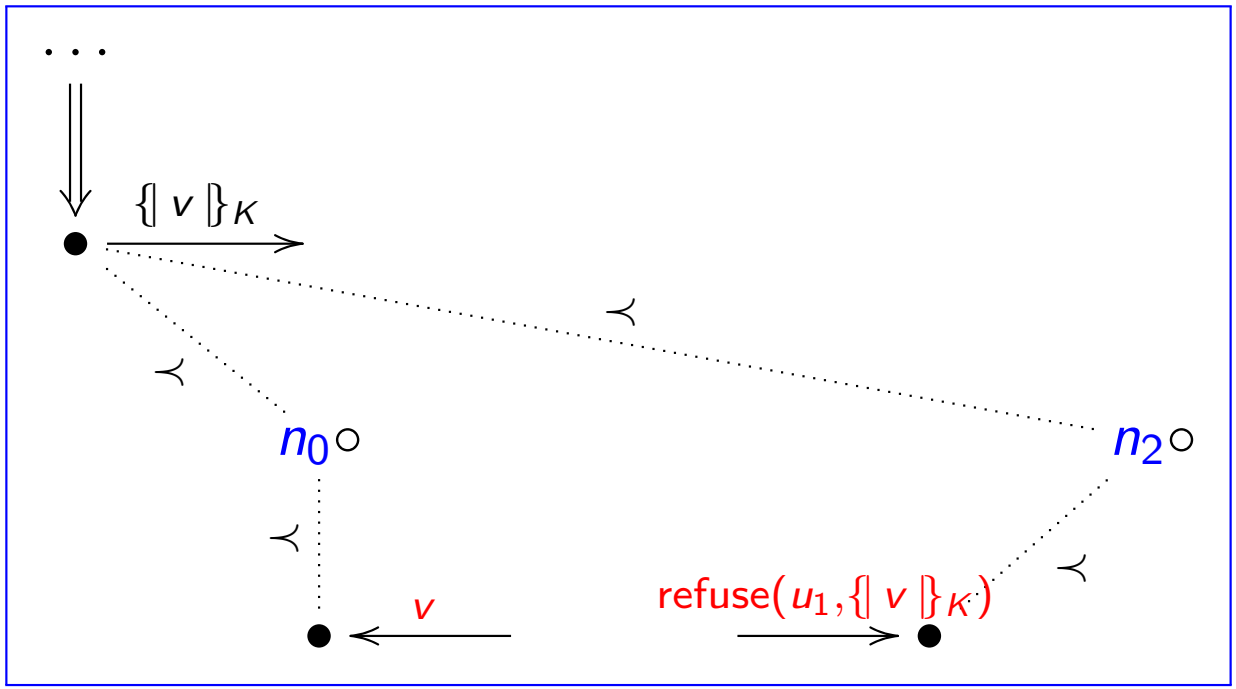
Security Goal, and consequence

This diagram never occurs within any execution:



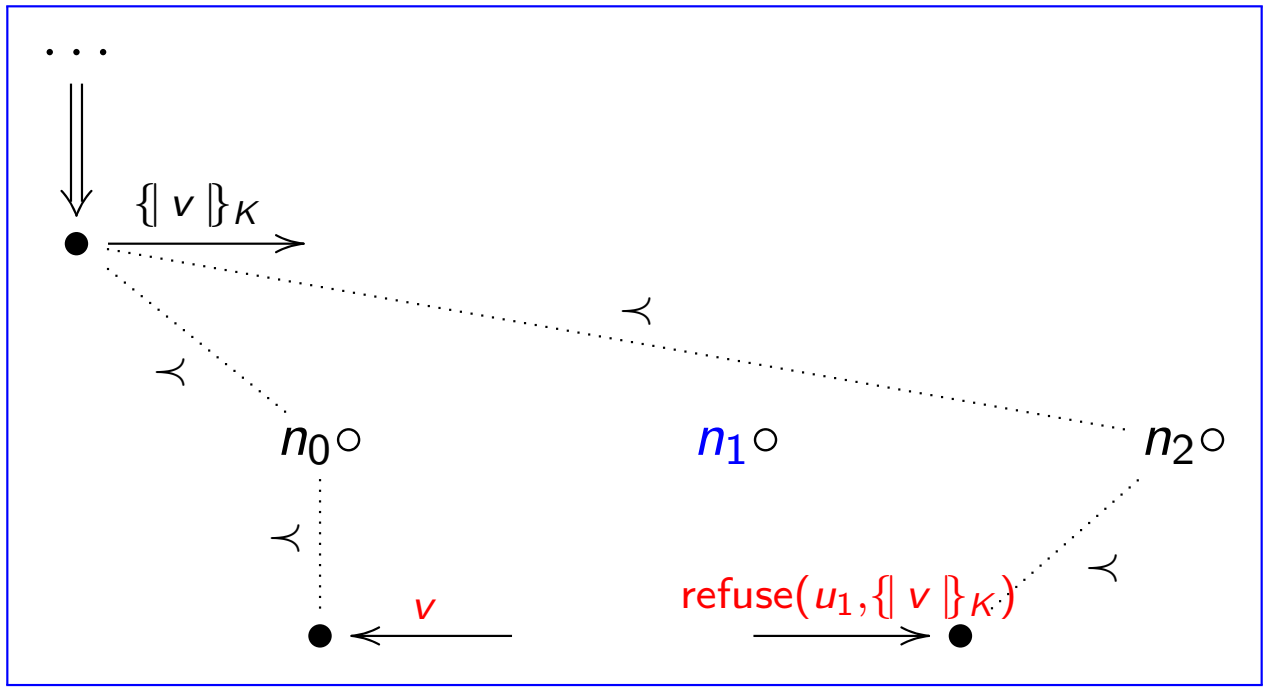
Security Goal, and consequence, 1

This diagram never occurs within any execution:



Security Goal, and consequences, 2

This diagram never occurs within any execution:



$$\exists n_1 \,.\, \mathrm{Boot}(n_1) \text{ and}$$
$$n_0 \preceq n_1 \preceq n_2 \quad \text{or} \quad n_2 \preceq n_1 \preceq n_0$$

by the Prefix/Boot Lemma

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR regardless of this choice

Produce $u_1 = \text{Hash}(u_0, N)$ by extending some state u_0 with a fresh nonce N

Implementation Idea

Using a TPM

1. Use one TPM platform configuration register
2. Put it in a state u_1
3. Allow decryption if extended to $\text{Hash}(u_1, \text{OBTAIN})$
4. Generate refusal cert if extended to $\text{Hash}(u_1, \text{REFUSE})$
5. Ensure u_1 can never again be value of PCR regardless of this choice

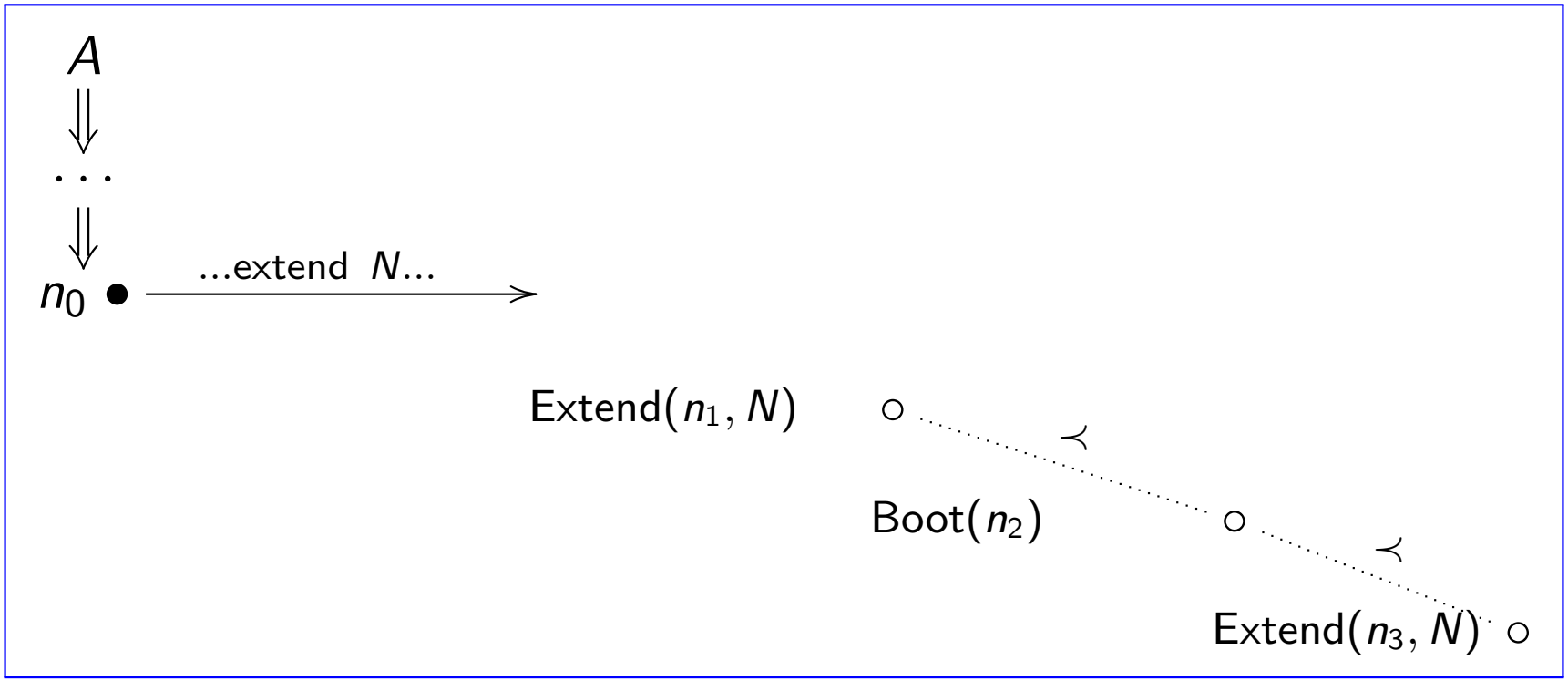
Produce $u_1 = \text{Hash}(u_0, N)$ by
extending some state u_0
with a fresh nonce N

Specification: PCR setup subprotocol, 1

This diagram never occurs within any execution:

N freshly chosen at n_0

N must be sent encrypted



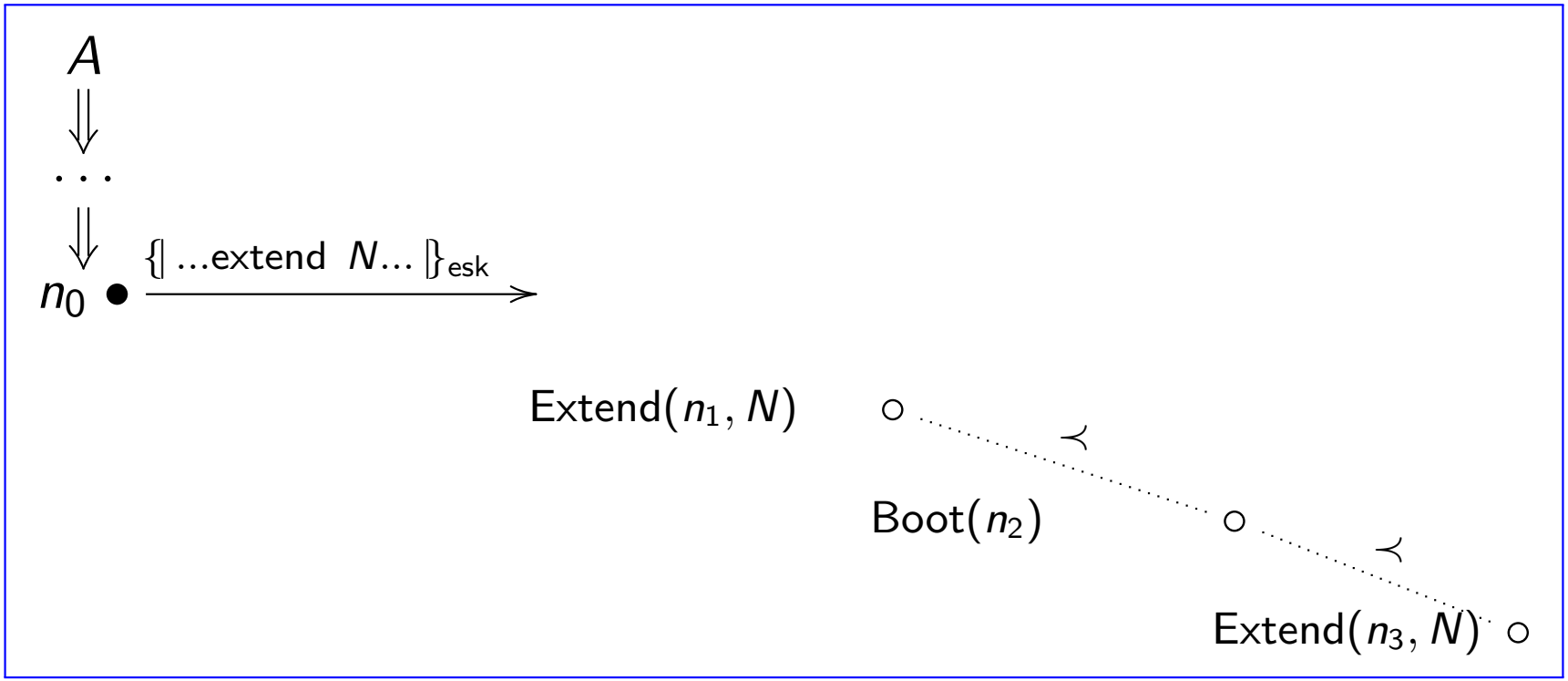
$\mathsf{pcr\_after}(n_1, \mathsf{Hash}(u_0, N))$

Specification: PCR setup subprotocol, 2

This diagram never occurs within any execution:

N freshly chosen at n_0

N must be sent encrypted



$\mathsf{pcr\_after}(n_1, \mathsf{Hash}(u_0, N))$

PCR Setup subprotocol

Implementation strategies

A delivers extend N to TPM
in an encrypted session
Then

either A closes the session before transmitting $\{\!|\, v \,|\!\}_K$

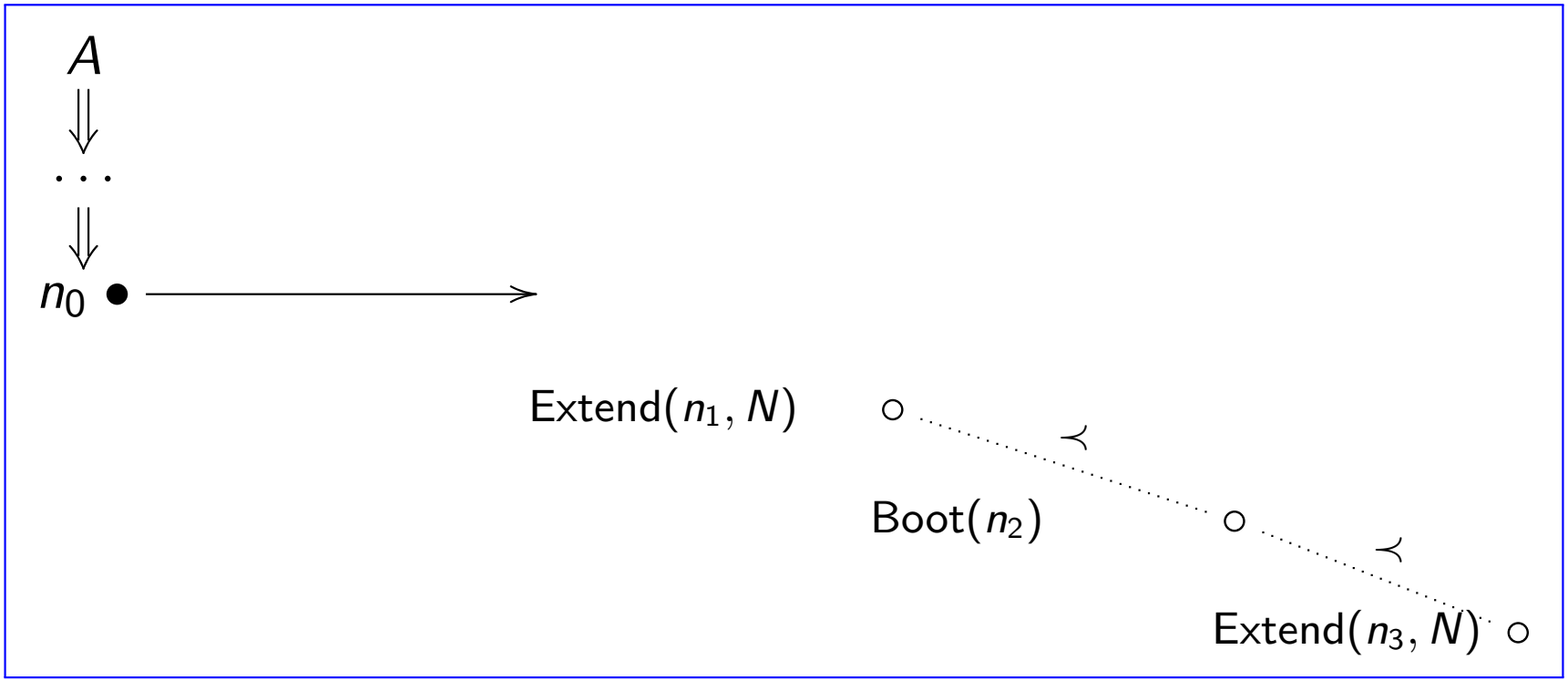
or TPM tears down encrypted sessions before reboot

Specification: PCR setup subprotocol, 2

This diagram never occurs within any execution:

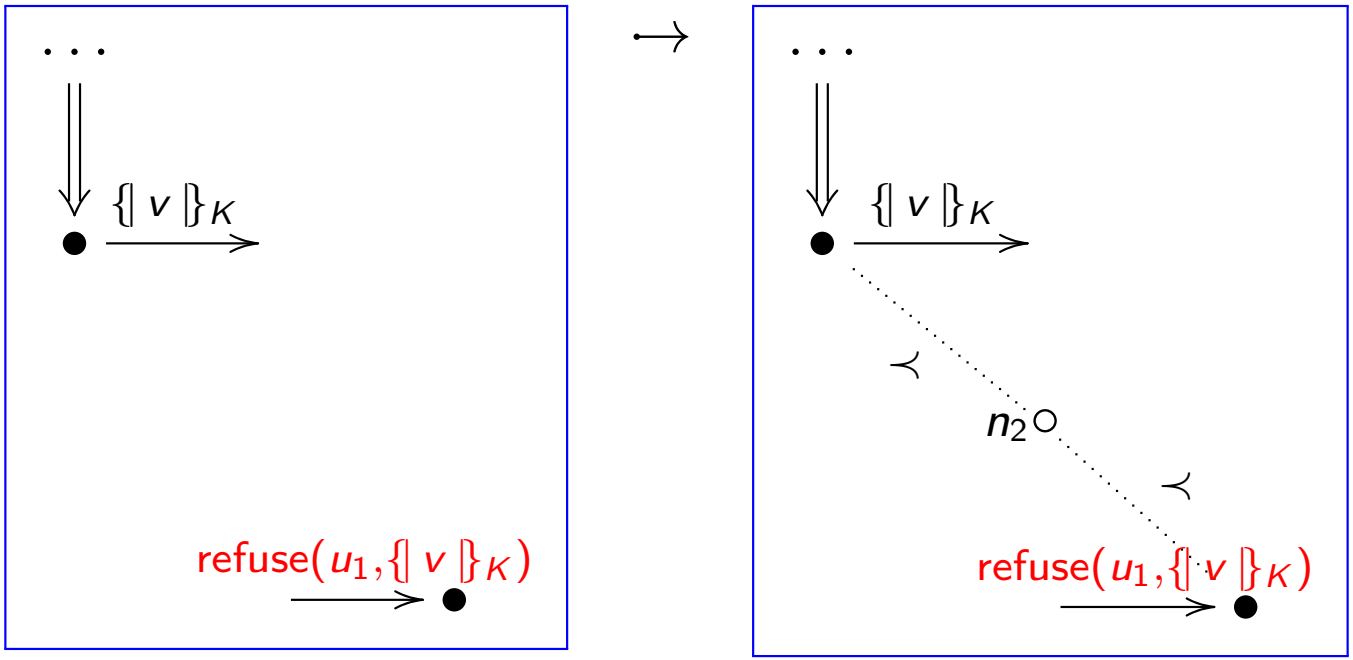
N freshly chosen at n_0

N must be sent encrypted



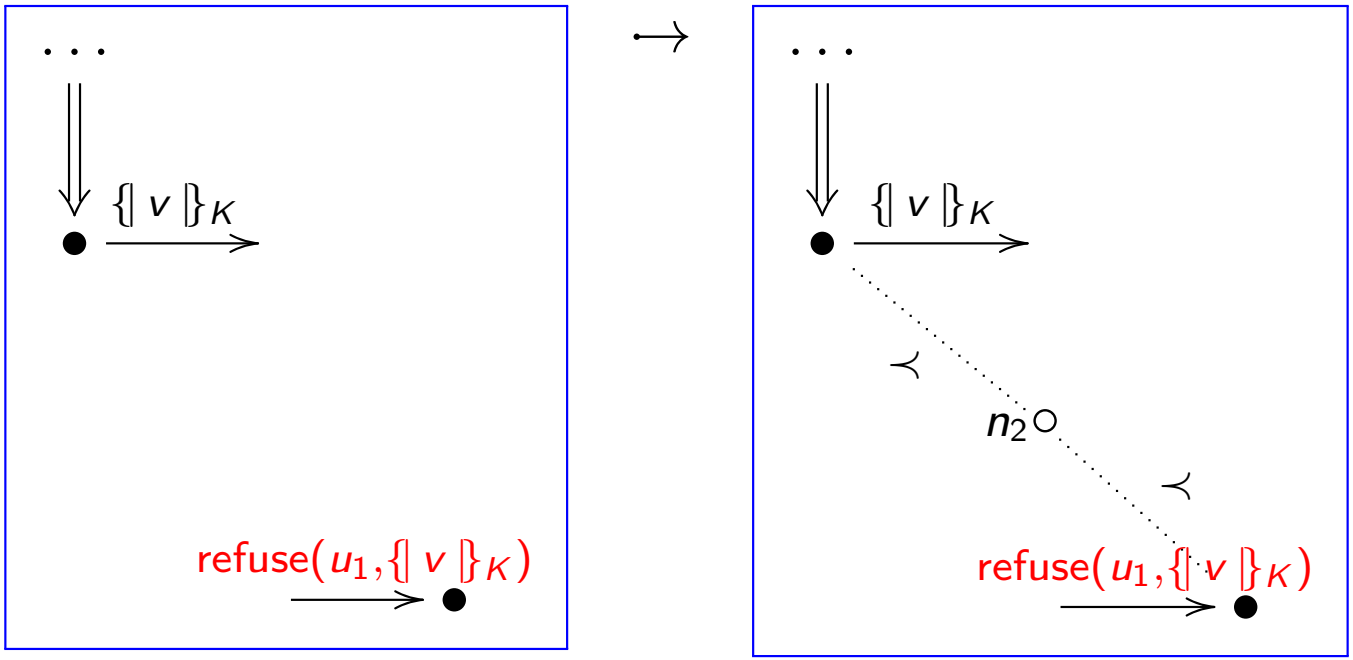
pcr_after(n_1, Hash(u_0, N))

Diagrams and Logic



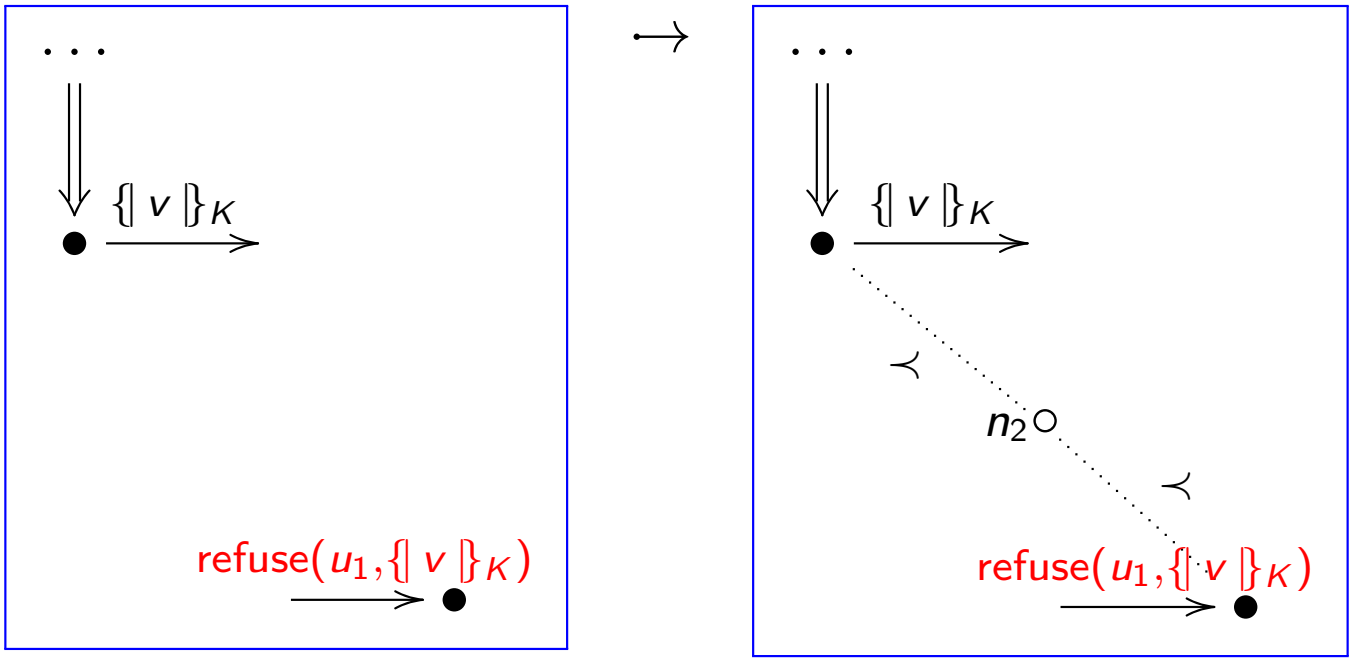
- Each box shows a structure (model)
- Positive existential formulas summarize structures
- Diagram says: Executions containing LHS contain RHS
- Homomorphisms preserve positive existential formulas

Diagrams and Logic



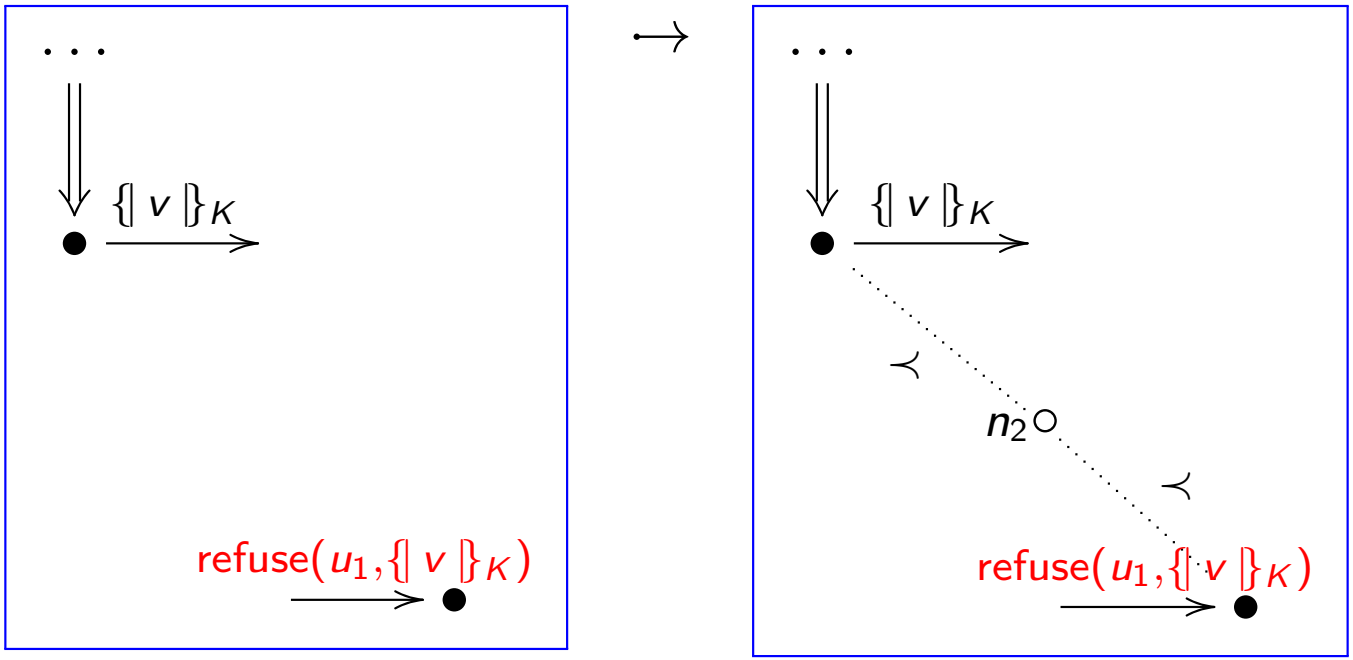
- Each box shows a structure (model)
- Positive existential formulas summarize structures
- Diagram says: Executions containing LHS contain RHS
- Homomorphisms preserve positive existential formulas

Diagrams and Logic



- Each box shows a structure (model)
- Positive existential formulas summarize structures
- Diagram says: Executions containing LHS contain RHS
- Homomorphisms preserve positive existential formulas

Diagrams and Logic



- Each box shows a structure (model)
- Positive existential formulas summarize structures
- Diagram says: Executions containing LHS contain RHS
- Homomorphisms preserve positive existential formulas