Mechanizing and Automating Cryptographic Arguments

ProTeCS: Proofs and Proof Techniques for Cryptographic Security

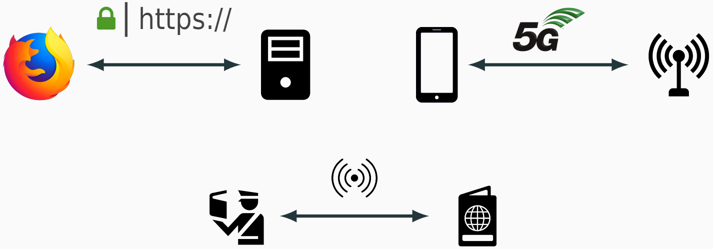
Adrien Koutsos Inria Paris

25 May 2024, Zurich

Context

Security Protocols

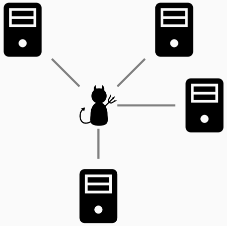
- **Distributed programs** which aim at providing some **security properties**.
- Uses **cryptographic primitives**: e.g. encryption.



Context: Attacker Model

Abstract Attacker Model

- **Network capabilities:** worst-case scenario: *eavesdrop*, *block* and *forge* messages.
- **Computational capabilities:** adversary is a Probabilistic Polynomial-time Turing Machine (PPTM).

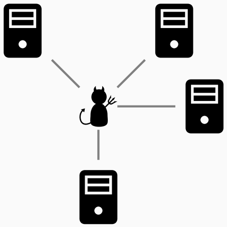


Attacks against security protocols can be very **damageable**, e.g. theft or privacy breach.

Context: Attacker Model

Abstract Attacker Model

- **Network capabilities:** worst-case scenario: *eavesdrop*, *block* and *forge* messages.
- **Computational capabilities:** adversary is a Probabilistic Polynomial-time Turing Machine (PPTM).



Attacks against security protocols can be very **damageable**, e.g. theft or privacy breach.

We need strong **security guarantees**.
$\Rightarrow$ can be provided by **cryptographic proofs**.

High-Confidence Security Guarantees

But security proofs are often **complicated** and **error-prone**:

- OAEP padding scheme:
  claimed secure in [BR94], **proof flawed** [Sho02].
- Fiat-Shamir with aborts:
  several proofs [Lyu12; KLS18] turned out to be **flawed** [Bar+23].
- several **logical attacks** on TLS, e.g.:
  TripleHandshake [Bha+14], LogJam [Adr+15].

High-Confidence Security Guarantees

But security proofs are often **complicated** and **error-prone**:

- OAEP padding scheme:
  claimed secure in [BR94], **proof flawed** [Sho02].
- Fiat-Shamir with aborts:
  several proofs [Lyu12; KLS18] turned out to be **flawed** [Bar+23].
- several **logical attacks** on TLS, e.g.:
  TripleHandshake [Bha+14], LogJam [Adr+15].

These are **critical** cryptographic designs under a lot of **public scrutiny**.
$\Rightarrow$ for such cryptographic designs, **manual proofs are insufficient**.

High-Confidence Security Guarantees

Verification for Cryptography
Formal mathematical proof of security protocols:

$$\begin{tikzcd}
\mathcal{S} \arrow[d, dashed] & \models \arrow[d, dashed] & \Phi \arrow[d, dashed] \\
\text{system} & \text{satisfies} & \text{property}
\end{tikzcd}$$

- **Machine-checked proofs** yield a high degree of confidence.
  - **general-purpose** tools (e.g. Coq and Lean).
  - in security protocol analysis, mostly **dedicated** tools.
    E.g. CryptoVerif, EasyCrypt, Squirrel.

Computer-aided Verification of Cryptographic Protocols

Goal
Design **formal frameworks** allowing for **mechanized verification** of **cryptographic arguments**.

- At the intersection of **cryptography** and **verification**.
- Particular verification challenges:
  - small or medium-sized programs
  - complex properties
  - probabilistic programs + arbitrary (resource-bounded) adversary

Mechanizing Cryptographic Proofs

Cryptographic Protocol Verification

Verification

$$\forall \text{😈} \in \mathcal{C}.\ \ (\text{😈} \mid\mid \mathcal{P}) \models \Phi$$

Requires a **formal framework** and **a tool** that can express:

- $\mathcal{P}$: the **protocol** under study.
- $\text{😈} \in \mathcal{C}$: the **adversarial model**, i.e. the class of adversaries.
- Φ: the **security property**.
- $\models$: the **cryptographic arguments**.

Cryptographic Protocol Verification

	computational model	EasyCrypt	Squirrel
$\mathcal{P}$	program	imperative program (sequential modeling)	pure program (execution trace modeled)
🐈‍⬛ $\in \mathcal{C}$	PPTM	abstract & stateful module A	uninterpreted pure function **att**$(\cdot)$
Φ	game	$\lvert \mathsf{Pr}(\mathcal{G}) \rvert \leq \epsilon$ $\lvert \mathsf{Pr}(\mathcal{G}) - \mathsf{Pr}(\mathcal{G}') \rvert \leq \epsilon$	$[\phi_{\mathcal{G}}]$ $\vec{u}_{\mathcal{G}} \sim \vec{u}_{\mathcal{G}'}$
$\models$	game-hops & reductions	program logics (pRHL)	probabilistic logics (CCSA)

Cryptographic Protocol Verification

	computational model	EasyCrypt	Squirrel
$\mathcal{P}$	program	imperative program (sequential modeling)	pure program (execution trace modeled)
🐈 $\in \mathcal{C}$	PPTM	abstract & stateful module A	uninterpreted pure function **att**$(\cdot)$
Φ	game	$\lvert \Pr(\mathcal{G}) \rvert \leq \epsilon$ $\lvert \Pr(\mathcal{G}) - \Pr(\mathcal{G}') \rvert \leq \epsilon$	$[\phi_{\mathcal{G}}]$ $\vec{u}_{\mathcal{G}} \sim \vec{u}_{\mathcal{G}'}$
$\models$	game-hops & reductions	program logics (pRHL)	probabilistic logics (CCSA)

- + expressive logics
- + can target implementations

- + temporal logic
- + higher-level rules
- + (usually) shorter proofs

The Squirrel Prover

Tool for the verification of **security protocols**:

- **Input language**: applied π-calculus.
- Implements a **CCSA probabilistic logic**:
  - **Reachability** properties: $[\phi_{\mathcal{G}}]$
  - **Indistinguishability** properties: $\vec{u}_{\mathcal{G}} \sim \vec{u}_{\mathcal{G}'}$
  - In the **asymptotic security** setting. E.g.

    $$\vec{u}_{\mathcal{G}} \sim \vec{u}_{\mathcal{G}'} \iff$$
    $$\forall 😈 \in \mathcal{C}.\ |\mathsf{Pr}(\mathcal{G}(😈)) - \mathsf{Pr}(\mathcal{G}'(😈))| \leq \epsilon_{\mathsf{negl}}$$

  - **Reasoning rules** valid w.r.t. any computational attacker 😈.

The Squirrel Prover

Proof assistant:

- Users prove goals using sequences of tactics.
  - **Generic maths.** tactics, e.g. apply, rewrite.
  - **Crypto.** tactics, e.g. cpa.
  - **Probabilistic** tactics, e.g. fresh.
  - **Structural** tactics, e.g. trans.
- Development done using a proof-general mode.

  As in Coq, EasyCrypt ...

The Squirrel Prover

Open-source tool

- Project web-page:

  `https://squirrel-prover.github.io/`

- Documentation web-page:

  `https://squirrel-prover.github.io/documentation/`

Mechanizing Cryptographic Proofs

The CCSA Framework

Formalizing Cryptographic Proofs

Our **formal framework** must model and capture:

- $\mathcal{P}$: **protocol**
- 😈 $\in \mathcal{C}$: **adversarial model**
- Φ: **security property**
- $\models$: **cryptographic arguments**

Limitations: what is not in this talk

- $\in \mathcal{C}$: **adversarial model**
  - in this talk: only **classical** adversaries, i.e. $\mathcal{C} = \mathrm{PPTM}$.
  - **quantum** adversaries (i.e. $\mathcal{C} = \mathrm{PQTM}$) are *work-in-progress*.

- Φ: **security property**
  - in this talk: **asymptotic** security.
  - there exists a **concrete** security version of the logic [CSF'24]
    (on paper, not implemented)

- $\models$: **cryptographic arguments**
  - standard **game-based** proofs.
  - other techniques may be out-of-scope:
    UC, rewinding, GGM, . . .
  - mechanizing crypto. proofs takes **time**:
    your favorite, complicated, crypto. designs may be difficult to formalize.

Modeling

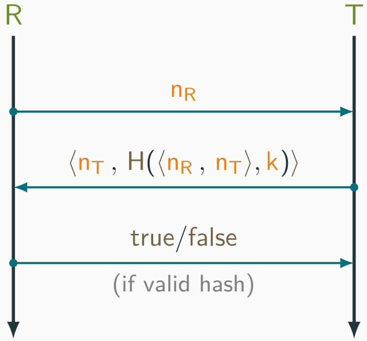
$$\forall \text{😈} \in \mathcal{C}.\ \ (\text{😈} \mid\mid \mathcal{P}) \models \Phi$$

- **Protocol** $\mathcal{P}$: a **concrete** concurrent program.
  In Squirrel, described in the applied π-calculus.
- **Adversarial model** $\text{😈} \in \mathcal{C}$: an **abstract** (i.e. unknown) PPTM program.
- **Full system** $=$ interaction $(\text{😈} \mid\mid \mathcal{P})$.

Example: The Hash-Lock Protocol

A simple example

- Two party **authentication protocol**: reader R $\Longleftrightarrow$ RFID tag T.
- Keyed-hash function H with a shared key k.



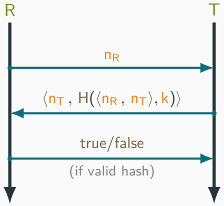
The Hash-Lock Protocol

In the **applied π-calculus:**

Hash-Lock

$$
\begin{aligned}
\mathsf{T}(\mathtt{i}) :\ & \textbf{input}(\mathsf{in}). \\
& \nu\, \mathsf{n}_{\mathsf{T},\mathtt{i}}. \\
& \textbf{let } \mathsf{h} = \mathsf{H}(\langle \mathsf{in}, \mathsf{n}_{\mathsf{T},\mathtt{i}} \rangle, \mathsf{k}) \textbf{ in} \\
& \textbf{let } \mathsf{out} = \langle \mathsf{n}_{\mathsf{T},\mathtt{i}}, \mathsf{h} \rangle \textbf{ in} \\
& \textbf{output}(\mathsf{out})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{R}(\mathtt{j}) :\ & \nu\, \mathsf{n}_{\mathsf{R},\mathtt{j}}. \\
& \textbf{output}(\mathsf{n}_{\mathsf{R},\mathtt{j}}). \\
& \textbf{input}(\mathsf{in}). \\
& \textbf{output}\big(\pi_2(\mathsf{in}) = \mathsf{H}(\langle \mathsf{n}_{\mathsf{R},\mathtt{j}}, \pi_1(\mathsf{in}) \rangle, \mathsf{k})\big)
\end{aligned}
$$



Modeling

How do we model the interaction (😈 $||$ $\mathcal{P}$) in a **pure language**?

$\Longrightarrow$ remove all **stateful** effects:

- **network I/O.**
- **random samplings.**

Modeling: Network I/O

I/O effects

- Network input $\Rightarrow$ function call to 😈.

For a **single I/O block** $\mathsf{T}(\mathtt{i})$:

```
input(in)
ν n_{T,i}
h := H(⟨in , n_{T,i}⟩, k)
out := ⟨n_{T,i} , h⟩
output(out)
```

$\longrightarrow$

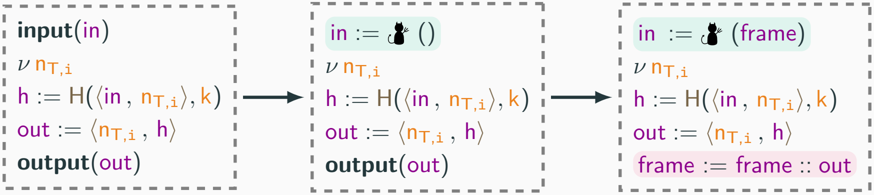
```
in := 😈 ()
ν n_{T,i}
h := H(⟨in , n_{T,i}⟩, k)
out := ⟨n_{T,i} , h⟩
output(out)
```

Modeling: Network I/O

I/O effects

- Network input $\Rightarrow$ function call to 🐈‍⬛.
- Network output $\Rightarrow$ add to 🐈‍⬛'s knowledge.

For a **single I/O block** $\mathtt{T(i)}$:



```
input(in)
ν n_{T,i}
h := H(⟨in, n_{T,i}⟩, k)
out := ⟨n_{T,i}, h⟩
output(out)
```

```
in := 🐈‍⬛ ()
ν n_{T,i}
h := H(⟨in, n_{T,i}⟩, k)
out := ⟨n_{T,i}, h⟩
output(out)
```

```
in := 🐈‍⬛ (frame)
ν n_{T,i}
h := H(⟨in, n_{T,i}⟩, k)
out := ⟨n_{T,i}, h⟩
frame := frame :: out
```

Modeling: Random Sampling

Probabilistic effects

- Move to an **early-sampling semantics** with indexed **names**:
  - name n_{T} is an array of **i.i.d. random samplings**.
  - random sampling $\nu\, \mathsf{n}_{\mathsf{T},\mathtt{i}} \Longrightarrow$ array access $\mathsf{n}_{\mathsf{T}}(\mathtt{i})$.

I/O block $\mathsf{T}(\mathtt{i})$:

$\mathsf{in} :=$ 🐈(frame)
$\nu\, \mathsf{n}_{\mathsf{T},\mathtt{i}}$
$\mathsf{h} := \mathsf{H}(\langle \mathsf{in}\,,\, \mathsf{n}_{\mathsf{T},\mathtt{i}} \rangle, \mathsf{k})$
$\mathsf{out} := \langle \mathsf{n}_{\mathsf{T},\mathtt{i}}\,,\, \mathsf{h} \rangle$
$\mathsf{frame} := \mathsf{frame} :: \mathsf{out}$

$\longrightarrow$

$\mathsf{in} :=$ 🐈(frame)
$\mathsf{h} := \mathsf{H}(\langle \mathsf{in}\,,\, \mathsf{n}_{\mathsf{T}}(\mathtt{i}) \rangle, \mathsf{k})$
$\mathsf{out} := \langle \mathsf{n}_{\mathsf{T}}(\mathtt{i})\,,\, \mathsf{h} \rangle$
$\mathsf{frame} := \mathsf{frame} :: \mathsf{out}$

Modeling: Execution Trace

Single I/O blocks:

$\mathsf{T}(\mathtt{i})$

in := 🐈‍⬛(frame)
$\mathsf{h} := \mathsf{H}(\langle \mathsf{in}\, , \mathsf{n_T}(\mathtt{i}) \rangle, \mathsf{k})$
$\mathsf{out} := \langle \mathsf{n_T}(\mathtt{i})\, , \mathsf{h} \rangle$
$\mathsf{frame} := \mathsf{frame} :: \mathsf{out}$

$\mathsf{R}_1(\mathtt{j})$

$\mathsf{out} := \mathsf{n_R}(\mathtt{j})$
$\mathsf{frame} := \mathsf{frame} :: \mathsf{out}$

$\mathsf{R}_2(\mathtt{j})$

in := 🐈‍⬛(frame)
$\mathsf{out} := \big(\pi_2(\mathsf{in}) = \mathsf{H}(\langle \mathsf{n_R}(\mathtt{j})\, , \pi_1(\mathsf{in}) \rangle, \mathsf{k})\big)$
$\mathsf{frame} := \mathsf{frame} :: \mathsf{out}$

Many I/O blocks, add the **time**:

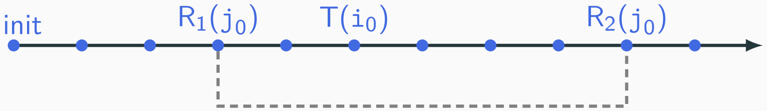
- index: type of **session numbers**.
- timestamp: type of **time-points** in an execution trace.

$$\tau ::= \mathsf{init} \mid \mathsf{T}(\mathtt{i}) \mid \mathsf{R}_1(\mathtt{i}) \mid \mathsf{R}_2(\mathtt{i}) \qquad (\text{where } \mathtt{i} : \mathsf{index})$$

Modeling: Execution Trace

Execution trace: timestamp $+$ order $<$.

Example:



Protocol execution encoded by **mutually recursive functions**:

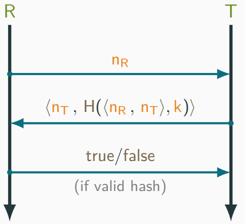
- $\mathtt{in}@\tau$: input at time τ
- $\mathtt{out}@\tau$: output at time τ
- $\mathtt{frame}@\tau$: 's knowledge at time τ, i.e. all $\mathtt{out}@\tau_0$ for $\tau_0 \leq \tau$.

Modeling: Execution Trace

$$
\begin{aligned}
\mathsf{in@}\tau \;&=\; \mathsf{match}\ \tau\ \mathsf{with}\\
&\qquad \mid \mathsf{init} \to \mathsf{empty}\\
&\qquad \mid \_ \to \text{😈}(\,\mathsf{frame@pred}(\tau)\,)\\[1em]
\mathsf{frame@}\tau \;&=\; \mathsf{match}\ \tau\ \mathsf{with}\\
&\qquad \mid \mathsf{init} \to \mathsf{empty}\\
&\qquad \mid \_ \to \mathsf{frame@pred}(\tau) :: \mathsf{out@}\tau
\end{aligned}
$$

Modeling: Execution Trace

$$
\begin{aligned}
\mathsf{in}@\tau \;&=\; \text{match } \tau \text{ with}\\
&\quad \mid \mathsf{init} \to \mathsf{empty}\\
&\quad \mid \_ \to \text{🐈‍⬛}(\,\mathsf{frame}@\mathsf{pred}(\tau)\,)\\[1em]
\mathsf{frame}@\tau \;&=\; \text{match } \tau \text{ with}\\
&\quad \mid \mathsf{init} \to \mathsf{empty}\\
&\quad \mid \_ \to \mathsf{frame}@\mathsf{pred}(\tau) :: \mathsf{out}@\tau\\[1em]
\mathsf{out}@\tau \;&=\; \text{match } \tau \text{ with}\\
&\quad \mid \mathsf{init} \to \mathsf{empty}\\
&\quad \mid \mathsf{T}(\mathtt{i}) \to \langle \mathsf{n_T}(\mathtt{i})\,,\, \mathsf{H}(\langle \mathsf{in}@\tau\,,\, \mathsf{n_T}(\mathtt{i})\rangle, \mathsf{k})\rangle\\
&\quad \mid \mathsf{R_1}(\mathtt{j}) \to \mathsf{n_R}(\mathtt{j})\\
&\quad \mid \mathsf{R_2}(\mathtt{j}) \to \pi_2(\mathsf{in}@\tau) = \mathsf{H}(\langle \mathsf{n_R}(\mathtt{j})\,,\, \pi_1(\mathsf{in}@\tau)\rangle, \mathsf{k})
\end{aligned}
$$



The CCSA Logic: Terms

Core Syntax

A higher-order λ-calculus with library, **adversarial** and recursive functions; names (for random samplings); and variables.

$$\begin{aligned} t &::= s \mid (t\ t) \mid \lambda(\mathsf{x} : \tau).\, t \\ s &\in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{😈} \in \mathcal{F}_{\mathsf{adv}}\} \cup \{m \in \mathcal{F}_{\mathsf{rec}}\} \cup \{\mathsf{n} \in \mathcal{N}\} \cup \{\mathsf{x} \in \mathcal{X}\} \end{aligned}$$

The CCSA Logic: Terms

Core Syntax

A higher-order λ-calculus with library, **adversarial** and recursive functions; names (for random samplings); and variables.

$$
\begin{aligned}
t &::= s \mid (t\ t) \mid \lambda(\mathsf{x} : \tau).\, t \\
s &\in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{😈} \in \mathcal{F}_{\mathsf{adv}}\} \cup \{m \in \mathcal{F}_{\mathsf{rec}}\} \cup \{\mathsf{n} \in \mathcal{N}\} \cup \{\mathsf{x} \in \mathcal{X}\}
\end{aligned}
$$

Types

$(t : \tau)$ is the type τ of term t:

- a base type, e.g.

  $\mathsf{bool} : \{\mathrm{true}, \mathrm{false}\}$ $\mathsf{message} : \{0, 1\}^*$ $\mathsf{int} : \mathbb{N}$

  timestamp : time-points index : session numbers

- an arrow type $\tau_0 \to \tau_1$, tuple type $\tau_0 * \tau_1$, . . .

The CCSA Logic: Terms

Core Syntax

A higher-order λ-calculus with library, **adversarial** and recursive functions; names (for random samplings); and variables.

$$\begin{array}{lll} t & ::= & s \mid (t\ t) \mid \lambda(\mathsf{x} : \tau).\, t \mid (t, \ldots, t) \mid \forall(\mathsf{x} : \tau).\, t \mid \text{match } t \text{ with } \ldots \\ s & \in & \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{😈} \in \mathcal{F}_{\mathsf{adv}}\} \cup \{m \in \mathcal{F}_{\mathsf{rec}}\} \cup \{\mathsf{n} \in \mathcal{N}\} \cup \{\mathsf{x} \in \mathcal{X}\} \end{array}$$

Types

$(t : \tau)$ is the type τ of term t:

- a base type, e.g.

  $\mathsf{bool} : \{\mathsf{true}, \mathsf{false}\}$ $\mathsf{message} : \{0, 1\}^*$ $\mathsf{int} : \mathbb{N}$

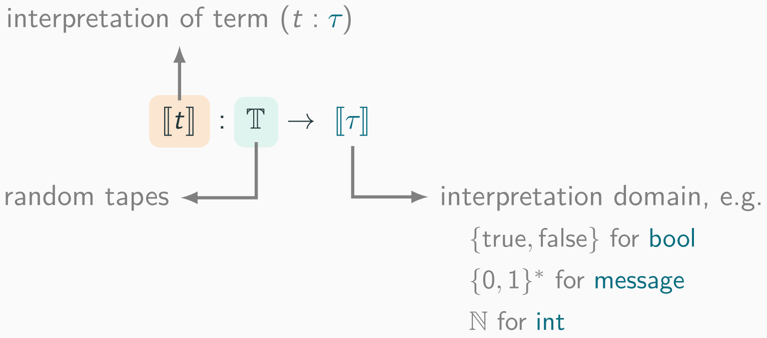
  timestamp : time-points index : session numbers

- an arrow type $\tau_0 \to \tau_1$, tuple type $\tau_0 * \tau_1$, ...

The CCSA Logic: Terms

The **semantics $[\![t]\!]$** uses **discrete random variables**, not **distributions**!

Shared source of randomness: **set of random tapes $\mathbb{T}$**.


interpretation of term $(t : \tau)$
$[\![t]\!] : \mathbb{T} \to [\![\tau]\!]$
random tapes
interpretation domain, e.g.
$\{\mathsf{true}, \mathsf{false}\}$ for bool
$\{0,1\}^*$ for message
$\mathbb{N}$ for int

Allow **probabilistic dependencies** between terms.

The CCSA Logic: Terms

Examples

- If $(\mathsf{n}, \mathsf{n}_0 : \mathsf{message})$ then:

$$\begin{aligned}
[\![\mathsf{n}]\!] &\approx \text{sample } w \text{ in } \{0,1\}^\eta \\
[\![(\mathsf{n}, \mathsf{n}_0)]\!] &\approx \text{sample } w \text{ in } \{0,1\}^\eta \\
& \text{sample } w' \text{ in } \{0,1\}^\eta \textbf{ independently} \\
& \text{build } (w, w') \\
[\![(\mathsf{n}, \mathsf{n})]\!] &\approx \text{sample } w \text{ in } \{0,1\}^\eta \\
& \text{build } (w, w)
\end{aligned}$$

$$[\![(\mathsf{n}, \mathsf{n})]\!] = ([\![\mathsf{n}]\!], [\![\mathsf{n}]\!]) = (w, w)$$

The CCSA Logic: Terms

Semantics

Standard semantics $[\![t]\!]_{\mathbb{M}}^{\eta,\rho} \in [\![\tau]\!]_{\mathbb{M}}$ parameterized by:

- the model $\mathbb{M}$.
- the **security parameter** η.
- a pair $\rho = (\rho_h, \rho_a)$ of **random tapes** $\rho \in \mathbb{T}_{\mathbb{M}}^{\eta}$:
  ρ_h for *honest* randomness, ρ_a for the adversary.
  (tapes ρ_h, ρ_a must be finite.)

The CCSA Logic: Terms

Semantics

Standard semantics $[\![t]\!]^{\eta,\rho}_{\mathbb{M}} \in [\![\tau]\!]_{\mathbb{M}}$ parameterized by:

- the model $\mathbb{M}$.
- the **security parameter** η.
- a pair $\rho = (\rho_h, \rho_a)$ of **random tapes** $\rho \in \mathbb{T}^{\eta}_{\mathbb{M}}$:
  ρ_h for *honest* randomness, ρ_a for the adversary.
  (tapes ρ_h, ρ_a must be finite.)

$$[\![\mathsf{f}(t)]\!]^{\eta,\rho}_{\mathbb{M}} \overset{\text{def}}{=} \mathbb{M}_{\mathsf{f}}(\eta, [\![t]\!]^{\eta,\rho}_{\mathbb{M}})$$

$$[\![\mathsf{n}(t)]\!]^{\eta,\rho}_{\mathbb{M}} \overset{\text{def}}{=} \mathbb{M}_{\mathsf{n}}(\eta, \rho_h, [\![t]\!]^{\eta,\rho}_{\mathbb{M}})$$

$$[\![\text{😈}(t)]\!]^{\eta,\rho}_{\mathbb{M}} \overset{\text{def}}{=} \mathbb{M}_{\text{😈}}(\eta, \rho_a, [\![t]\!]^{\eta,\rho}_{\mathbb{M}})$$

Machines $\mathbb{M}_{\mathsf{f}}, \mathbb{M}_{\mathsf{n}}, \mathbb{M}_{\text{😈}}$ are deterministic
ptime (w.r.t. η + size of the args.)

The CCSA Logic: Terms

Names

- Take n : index $\rightarrow$ message.

  n(i): uniform random samplings over bit-strings of length η

The CCSA Logic: Terms

Names

- Take n : index $\rightarrow$ message.
  $\mathsf{n}(i)$: uniform random samplings over bit-strings of length η
- ($\neq$ name symbols or $\neq$ indices) $\Longrightarrow$ **independent** samplings.
  Thus:
  $$\Pr_\rho([\![\mathsf{n}_0(i_0)]\!]^{\eta,\rho} = [\![\mathsf{n}_1(i_1)]\!]^{\eta,\rho}) = \frac{1}{2^\eta}$$
  if $\mathsf{n}_0 \neq \mathsf{n}_1$ or if ($[\![i_0 \neq i_1]\!]^{\eta,\rho}$ for all η, ρ).

The CCSA Logic: Terms

Names

- Take n : index → message.

  n(i): uniform random samplings over bit-strings of length η

- ($\neq$ name symbols or $\neq$ indices) $\Longrightarrow$ **independent** samplings.

  Thus:

$$\Pr_\rho(\llbracket \mathsf{n}_0(i_0)\rrbracket^{\eta,\rho} = \llbracket \mathsf{n}_1(i_1)\rrbracket^{\eta,\rho}) = \frac{1}{2^\eta}$$

  if $\mathsf{n}_0 \neq \mathsf{n}_1$ or if ($\llbracket i_0 \neq i_1 \rrbracket^{\eta,\rho}$ for all η, ρ).

- Going further, if m does not occur in t:

$$\Pr_\rho(\llbracket \mathsf{m} = t\rrbracket^{\eta,\rho}) = \frac{1}{2^\eta}$$

  For now, "m does not occur in t" means
  t without recursive functions + $\mathsf{m} \notin \mathrm{st}(t)$.

The CCSA Logic: Terms

- The logic has a **standard semantics**,
- but a **particular** interpretation domain.

$$[\![t]\!]_{\mathbb{M}}^{\eta,\rho} \in [\![\tau]\!]_{\mathbb{M}} \qquad \Longrightarrow \qquad [\![t]\!]_{\mathbb{M}} \in \mathbb{RV}_{\mathbb{M}}(\tau)$$

$\mathbb{RV}_{\mathbb{M}}(\tau)$: η-families of random-variables over $[\![\tau]\!]_{\mathbb{M}}$.

$$\mathbb{RV}_{\mathbb{M}}(\tau) = \Big(\mathbb{T}_{\mathbb{M}}^{\eta} \to [\![\tau]\!]_{\mathbb{M}} \Big)_{\eta \in \mathbb{N}}$$

Formalizing Cryptographic Proofs

Our **formal framework** must model and capture:

- $\mathcal{P}$: **protocol** ✓
- 😈 $\in \mathcal{C}$: **adversarial model** ✓
- Φ: **security property**
- $\models$: **cryptographic arguments**

The CCSA Logic: Security Predicates

We consider two main **security predicates**:

- $[\phi]$: the term ϕ of type bool is **overwhelmingly true**:

$$\mathbb{M} \models [\phi] \quad \text{iff.} \quad \mathsf{Pr}_\rho\left(\llbracket\phi\rrbracket_{\mathbb{M}}^{\eta,\rho}\right) \text{ negligible in } \eta.$$

The CCSA Logic: Security Predicates

We consider two main **security predicates**:

- $[\phi]$: the term ϕ of type bool is **overwhelmingly true**:

$$\mathbb{M} \models [\phi] \quad \text{iff.} \quad \mathsf{Pr}_\rho\left(\llbracket \phi \rrbracket_{\mathbb{M}}^{\eta,\rho}\right) \text{ negligible in } \eta.$$

- $\vec{u}_0 \sim \vec{u}_1$: $\vec{u}_0$ and $\vec{u}_1$ are **indistinguishable**:

$$\mathbb{M} \models \vec{u}_0 \sim \vec{u}_1 \text{ iff. } \forall 😈 \in \mathcal{C}. \left| \begin{array}{l} \quad \mathsf{Pr}_\rho\left(😈(\eta, \llbracket \vec{u}_0 \rrbracket_{\mathbb{M}}^{\eta,\rho}, \rho_a)\right) \\ - \ \mathsf{Pr}_\rho\left(😈(\eta, \llbracket \vec{u}_1 \rrbracket_{\mathbb{M}}^{\eta,\rho}, \rho_a)\right) \end{array} \right| \text{ negligible in } \eta$$

The CCSA Logic: Security Predicates

We consider two main **security predicates**:

- $[\phi]$: the term ϕ of type bool is **overwhelmingly true**:

$$\mathbb{M} \models [\phi] \quad \text{iff.} \quad \mathsf{Pr}_\rho\left(\llbracket \phi \rrbracket_{\mathbb{M}}^{\eta,\rho}\right) \text{ negligible in } \eta.$$

- $\vec{u}_0 \sim \vec{u}_1$: $\vec{u}_0$ and $\vec{u}_1$ are **indistinguishable**:

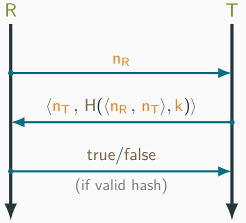
$$\mathbb{M} \models \vec{u}_0 \sim \vec{u}_1 \text{ iff. } \forall \text{😈} \in \mathcal{C}. \left| \begin{array}{l} \quad \mathsf{Pr}_\rho\left(\text{😈}(\eta, \llbracket \vec{u}_0 \rrbracket_{\mathbb{M}}^{\eta,\rho}, \rho_a)\right) \\ - \; \mathsf{Pr}_\rho\left(\text{😈}(\eta, \llbracket \vec{u}_1 \rrbracket_{\mathbb{M}}^{\eta,\rho}, \rho_a)\right) \end{array} \right| \text{ negligible in } \eta$$

$$\left(\begin{array}{l} \vec{u}_0 = t_1, \ldots, t_n \\ \vec{u}_1 = s_1, \ldots, s_n \end{array} \text{ and } t_i \text{ and } s_i \text{ have the same type } \forall i \right)$$

The CCSA Logic: Security Predicates

Authentication for Hash-Lock:

$$\left[\begin{array}{l}(\mathsf{out}@\mathsf{R}_2(j) = \mathsf{true}) \Rightarrow \\ \qquad \exists i : \mathsf{index}. \quad \mathsf{R}_1(j) < \mathsf{T}(i) < \mathsf{R}_2(j) \\ \qquad\qquad \wedge\ \mathsf{out}@\mathsf{R}_1(j) = \mathsf{in}@\mathsf{T}(i) \\ \qquad\qquad \wedge\ \mathsf{out}@\mathsf{T}(i) = \mathsf{in}@\mathsf{R}_2(j)\end{array}\right]$$



Weak **privacy** for Hash-Lock:

$$\begin{array}{ll} & \mathsf{frame}@\mathsf{pred}(\mathsf{T}(i)), \mathsf{H}(\langle \mathsf{in}@\mathsf{T}(i) , \mathsf{n_T}(i)\rangle, \mathsf{k}) \\ \sim & \mathsf{frame}@\mathsf{pred}(\mathsf{T}(i)), \mathsf{n_{fresh}}\end{array}$$

The CCSA Logic: Global Logic

Squirrel's has **two kinds of formulas**:

- **Local formulas** are terms of type bool (e.g. $\phi_0 \Rightarrow \exists x.\,(\phi_1 \wedge \phi_2)$).

$$\phi ::= \ \phi \wedge \phi \mid \neg\,\phi \mid \forall x.\,\phi \mid t = t \mid \ldots$$

The CCSA Logic: Global Logic

SQUIRREL's has **two kinds of formulas**:

- **Local formulas** are terms of type bool (e.g. $\phi_0 \Rightarrow \exists x.\,(\phi_1 \wedge \phi_2)$).

$$\phi ::= \ \phi \wedge \phi \mid \neg\, \phi \mid \forall x.\, \phi \mid t = t \mid \ldots$$

- **Global formulas:** $\mathsf{FO}([\,\cdot\,], \cdot \sim \cdot, \ldots)$.

$$F ::= \qquad [\phi] \mid \vec{t} \sim \vec{t}$$

The CCSA Logic: Global Logic

SQUIRREL's has **two kinds of formulas**:

- **Local formulas** are terms of type bool (e.g. $\phi_0 \Rightarrow \exists x.\, (\phi_1 \wedge \phi_2)$).

$$\phi ::= \ \phi \wedge \phi \mid \neg\, \phi \mid \forall x.\, \phi \mid t = t \mid \ldots$$

- **Global formulas:** $\mathsf{FO}([\,\cdot\,], \cdot \sim \cdot, \ldots)$.

$$F ::= \ F \mathrel{\tilde{\wedge}} F \mid \tilde{\neg}\, F \mid \tilde{\forall} x.\, F \mid [\phi] \mid \vec{t} \sim \vec{t} \mid \mathsf{const}(t) \mid \ldots$$

Global formulas are Squirrel's **ambient logic**.

The CCSA Logic: Global Logic

Semantics of the global logic

Standard FO semantics but particular interpretation domain $\mathbb{RV}_{\mathbb{M}}(\tau)$:

- $\tilde{\forall}(x : \tau)$ means "for all η-**family** of **random variable** x over $[\![\tau]\!]$"

$$\mathbb{M} \models \tilde{\forall}(x : \tau).\ F \qquad \text{iff.} \qquad \mathbb{M}\{x \mapsto X\} \models F \text{ for all } X \in \mathbb{RV}_{\mathbb{M}}(\tau)$$

The CCSA Logic: Global Logic

Examples of valid global formulas

- $[(\phi = \mathsf{true}) \vee (\phi = \mathsf{false})]$

The CCSA Logic: Global Logic

Examples of valid global formulas

- $[(\phi = \mathsf{true}) \vee (\phi = \mathsf{false})]$
- $(\phi \sim \mathsf{true}) \ \tilde{\Leftrightarrow}\ [\phi]$

The CCSA Logic: Global Logic

Examples of valid global formulas

- $[(\phi = \mathsf{true}) \vee (\phi = \mathsf{false})]$
- $(\phi \sim \mathsf{true}) \ \tilde{\Leftrightarrow} \ [\phi]$
- $\big([s = t] \ \tilde{\wedge} \ u\{s\} \sim v\big) \ \tilde{\Rightarrow} \ \big(u\{t\} \sim v\big)$

The CCSA Logic: Global Logic

Examples of valid global formulas

- $[(\phi = \text{true}) \vee (\phi = \text{false})]$
- $(\phi \sim \text{true}) \;\tilde{\Leftrightarrow}\; [\phi]$
- $\big([s = t] \;\tilde{\wedge}\; u\{s\} \sim v\big) \;\tilde{\Rightarrow}\; \big(u\{t\} \sim v\big)$
- $[u = v] \tilde{\Rightarrow} u \sim v$ but not the converse:
  e.g. $\mathsf{n}_0 \sim \mathsf{n}_1$ but $[\mathsf{n}_0 \neq \mathsf{n}_1]$

The CCSA Logic: Global Logic

Examples of valid global formulas

- $[(\phi = \mathsf{true}) \vee (\phi = \mathsf{false})]$
- $(\phi \sim \mathsf{true}) \;\tilde{\Leftrightarrow}\; [\phi]$
- $\big([s = t] \;\tilde{\wedge}\; u\{s\} \sim v\big) \;\tilde{\Rightarrow}\; (u\{t\} \sim v)$
- $[u = v] \tilde{\Rightarrow} u \sim v$ but not the converse:
  e.g. $\mathsf{n}_0 \sim \mathsf{n}_1$ but $[\mathsf{n}_0 \neq \mathsf{n}_1]$

$\sim$ is not compositional

$$(u_0 \sim u_1) \,\tilde{\wedge}\, (v_0 \sim v_1) \qquad \text{does not always implies} \qquad u_0, v_0 \sim u_1, v_1$$

Counter-example:

$$\mathsf{n}_0 \sim \mathsf{n}_0 \text{ and } \mathsf{n}_0 \sim \mathsf{n}_1 \text{ but } \mathsf{n}_0, \mathsf{n}_0 \not\sim \mathsf{n}_0, \mathsf{n}_1$$

The CCSA Logic: Global Logic

$\neq$ between local/global formulas

$$[\phi \wedge \psi] \overset{?}{\Leftrightarrow} [\phi] \mathrel{\tilde{\wedge}} [\psi]$$

$$[\phi \vee \psi] \overset{?}{\Leftrightarrow} [\phi] \mathrel{\tilde{\vee}} [\psi]$$

$$[\phi \Rightarrow \psi] \overset{?}{\Leftrightarrow} [\phi] \mathrel{\tilde{\Rightarrow}} [\psi]$$

The CCSA Logic: Global Logic

$\neq$ between local/global formulas

$$[\phi \wedge \psi] \quad \Leftrightarrow \quad [\phi] \mathrel{\tilde{\wedge}} [\psi]$$

$$[\phi \vee \psi] \quad \overset{?}{\Leftrightarrow} \quad [\phi] \mathrel{\tilde{\vee}} [\psi]$$

$$[\phi \Rightarrow \psi] \quad \overset{?}{\Leftrightarrow} \quad [\phi] \mathrel{\tilde{\Rightarrow}} [\psi]$$

The CCSA Logic: Global Logic

$\neq$ between local/global formulas

$$[\phi \wedge \psi] \quad \Leftrightarrow \quad [\phi] \mathrel{\tilde{\wedge}} [\psi]$$

$$[\phi \vee \psi] \quad \Leftarrow \quad [\phi] \mathrel{\tilde{\vee}} [\psi]$$

$$[\phi \Rightarrow \psi] \quad \overset{?}{\Leftrightarrow} \quad [\phi] \mathrel{\tilde{\Rightarrow}} [\psi]$$

Counter-example for $\vee / \tilde{\vee}$:

$[(b = \mathsf{true}) \vee (b = \mathsf{false})]$ — valid

$[b = \mathsf{true}] \mathrel{\tilde{\vee}} [b = \mathsf{false}]$ — not valid

The CCSA Logic: Global Logic

$\neq$ between local/global formulas

$$\begin{aligned}
[\phi \wedge \psi] &\;\Leftrightarrow\; [\phi] \,\tilde{\wedge}\, [\psi] \\
[\phi \vee \psi] &\;\Leftarrow\; [\phi] \,\tilde{\vee}\, [\psi] \\
[\phi \Rightarrow \psi] &\;\Rightarrow\; [\phi] \,\tilde{\Rightarrow}\, [\psi]
\end{aligned}$$

Counter-example for $\vee / \tilde{\vee}$:

$[(b = \mathsf{true}) \vee (b = \mathsf{false})]$	$[b = \mathsf{true}] \,\tilde{\vee}\, [b = \mathsf{false}]$
valid	not valid

Counter-example for $\Rightarrow / \tilde{\Rightarrow}$:

$[(\mathsf{n} = 0) \Rightarrow (\mathsf{n} = 1)]$	$[\mathsf{n} = 0] \,\tilde{\Rightarrow}\, [\mathsf{n} = 1]$
not valid	valid

The CCSA Logic: Global Logic

The global logic is used as **ambient logic**.

Authentication for Hash-Lock:

$$\left[\begin{array}{l} (\mathsf{out}@\mathsf{R}_2(j) = \mathsf{true}) \Rightarrow \\ \qquad \exists i : \mathsf{index}. \quad \mathsf{R}_1(j) < \mathsf{T}(i) < \mathsf{R}_2(j) \\ \qquad\qquad\qquad \wedge\ \mathsf{out}@\mathsf{R}_1(j) = \mathsf{in}@\mathsf{T}(i) \\ \qquad\qquad\qquad \wedge\ \mathsf{out}@\mathsf{T}(i) \ = \mathsf{in}@\mathsf{R}_2(j) \end{array}\right]$$

Weak **privacy** for Hash-Lock:

$$\begin{array}{ll} & \mathsf{frame}@\mathsf{pred}(\mathsf{T}(i)), \mathsf{H}(\langle \mathsf{in}@\mathsf{T}(i)\,,\, \mathsf{n}_\mathsf{T}(i)\rangle, \mathsf{k}) \\ \sim & \mathsf{frame}@\mathsf{pred}(\mathsf{T}(i)), \mathsf{n}_{\mathsf{fresh}} \end{array}$$

The CCSA Logic: Global Logic

The global logic is used as **ambient logic**.

Authentication for Hash-Lock:

$$\tilde{\forall}(j : \mathsf{index}).\ \mathsf{const}(j)\ \tilde{\Rightarrow} \left[\begin{array}{l} (\mathsf{out}@\mathsf{R}_2(j) = \mathsf{true}) \Rightarrow \\ \qquad \exists i : \mathsf{index}. \quad \mathsf{R}_1(j) < \mathsf{T}(i) < \mathsf{R}_2(j) \\ \qquad\qquad\qquad \wedge\ \mathsf{out}@\mathsf{R}_1(j) = \mathsf{in}@\mathsf{T}(i) \\ \qquad\qquad\qquad \wedge\ \mathsf{out}@\mathsf{T}(i) \ = \mathsf{in}@\mathsf{R}_2(j) \end{array}\right]$$

Weak **privacy** for Hash-Lock:

$$\tilde{\forall}(i : \mathsf{index}).\ \mathsf{const}(i)\ \tilde{\Rightarrow} \begin{array}{l} \quad \mathsf{frame}@\mathsf{pred}(\mathsf{T}(i)), \mathsf{H}(\langle \mathsf{in}@\mathsf{T}(i)\,,\ \mathsf{n}_{\mathsf{T}}(i)\rangle, \mathsf{k}) \\ \sim\ \mathsf{frame}@\mathsf{pred}(\mathsf{T}(i)), \mathsf{n}_{\mathsf{fresh}} \end{array}$$

Formalizing Cryptographic Proofs

Our **formal framework** must model and capture:

- $\mathcal{P}$: **protocol** ✓
- 😈 $\in \mathcal{C}$: **adversarial model** ✓
- Φ: **security property** ✓
- $\models$: **cryptographic arguments**

Cryptographic Arguments

High-level structure of a **game-hopping** proof:

$$\mathcal{G}_0 \sim_{\epsilon_1} \cdots \sim_{\epsilon_n} \mathcal{G}_n \quad \Rightarrow$$
$$\mathcal{G}_0 \sim_{\epsilon_1+\cdots+\epsilon_n} \mathcal{G}_n$$

where each step $\mathcal{G}_i \sim_{\epsilon_{i+1}} \mathcal{G}_{i+1}$ is justified by:

- a cryptographic reduction to some hardness assumption.
- up-to-bad argument $|\Pr(\mathcal{G}) - \Pr(\mathcal{G}')| \leq \Pr(\text{bad})$.
  - $\Pr(\text{bad}) \leq \epsilon$ through a probabilistic argument (e.g. collision probability).
  - ...
- bridging steps showing that $\mathcal{G} \sim_0 \mathcal{G}'$.

$\Longrightarrow$ how to **capture these arguments in the logic**?

The CCSA Logic: Reasoning Rules

High-level structure

Basic properties of indistinguishability:

$$\textsc{Trans}\ \frac{\vec{u} \sim \vec{w} \qquad \vec{w} \sim \vec{v}}{\vec{u} \sim \vec{v}} \qquad \textsc{Sym}\ \frac{\vec{v} \sim \vec{u}}{\vec{u} \sim \vec{v}} \qquad \textsc{Refl}\ \frac{}{\vec{u} \sim \vec{u}}$$

The CCSA Logic: Reasoning Rules

Bridging steps

Captured by our rewriting rule:

$$\frac{[s = t] \qquad \vec{u}\{t\} \sim \vec{v}}{\vec{u}\{s\} \sim \vec{v}}\ \textsc{Rewrite}$$

and generic mathematical reasoning to prove $[s = t]$.

E.g. **functional properties** can be stated as **axioms**:

$$[\forall m, k.\ \mathsf{sdec}(\mathsf{senc}(m, k), k) = m]$$

The CCSA Logic: Reasoning Rules

Up-to-bad arguments

Two games $\mathcal{G}, \mathcal{G}'$ such that:

$$\Pr(\mathcal{G} \wedge \neg \mathsf{bad}) = \Pr(\mathcal{G}' \wedge \neg \mathsf{bad}).$$

Then $|\Pr(\mathcal{G}) - \Pr(\mathcal{G}')| \leq \Pr(\mathsf{bad})$.

In the **CCSA** logic:

$$\frac{[\phi_{\mathsf{bad}}] \qquad [\neg \phi_{\mathsf{bad}} \Rightarrow \vec{u} = \vec{v}]}{\vec{u} \sim \vec{v}}\ \text{U2B}$$

(similar to the rewrite rule for overwhelmingly equalities.)

The CCSA Logic: Reasoning Rules

Up-to-bad arguments

Two games $\mathcal{G}, \mathcal{G}'$ such that:

$$\mathsf{Pr}(\mathcal{G} \wedge \neg \mathsf{bad}) = \mathsf{Pr}(\mathcal{G}' \wedge \neg \mathsf{bad}).$$

Then $|\,\mathsf{Pr}(\mathcal{G}) - \mathsf{Pr}(\mathcal{G}')| \leq \mathsf{Pr}(\mathsf{bad})$.

In the **CCSA** logic:

$$\frac{[\phi_{\mathsf{bad}}] \qquad [\neg\phi_{\mathsf{bad}} \Rightarrow \vec{u} = \vec{v}]}{\vec{u} \sim \vec{v}}\ \text{U2B}$$

(similar to the rewrite rule for overwhelmingly equalities.)

Other direction $[\,\cdot\,] \Rightarrow (\cdot \sim \cdot)$ also exists:

$$\frac{[\psi] \qquad \phi \sim \psi}{[\phi]}\ \textsc{Rewrite-Equiv}$$

$\Longrightarrow$ enables **back-and-forth between both predicates**.

The CCSA Logic: Reasoning Rules

Probabilistic reasoning: collision of random samplings

n a name of type message:

$$\textsc{Indep}\ \frac{}{[\mathsf{n} \neq t]} \qquad \text{if } \mathsf{n} \text{ does not occur in } t$$

The CCSA Logic: Reasoning Rules

Probabilistic reasoning: collision of random samplings

n a name of type message:

$$\textsc{Indep}\ \frac{}{[\mathsf{n} \neq t]} \qquad \text{if } \mathsf{n} \text{ does not occur in } t$$

How to check that n does not occur in t?

- no recursive functions: direct syntactic check.
  Example: $[\mathsf{n} \neq$ 😈$(\mathsf{n}_0)]$

The CCSA Logic: Reasoning Rules

Probabilistic reasoning: collision of random samplings

n a name of type message:

$$\textsc{Indep}\ \frac{}{[\mathsf{n} \neq t]} \qquad \text{if } \mathsf{n} \text{ does not occur in } t$$

How to check that n does not occur in t?

- no recursive functions: direct syntactic check.
  Example: $[\mathsf{n} \neq$ 😈$(\mathsf{n}_0)]$
- with recursive functions: check recursive function definitions.
  Example: $[\mathsf{n} \neq$ 😈$(\mathsf{frame}@\tau)]$

The CCSA Logic: Reasoning Rules

More complicated with **indexed names**, e.g. $\mathsf{n_R}(\mathtt{j_0}) \neq$ 🐈$(\mathsf{frame}@\tau)$.

$\Longrightarrow$ use **local formulas** to ensure freshness.

```
out@τ =
 match τ with
   | init → empty
   | T(i) → ⟨n_T(i), H(⟨in@τ, n_T(i)⟩, k)⟩
   | R_1(j) → n_R(j)
   | R_2(j) → π_2(in@τ) = H(⟨n_R(j), π_1(in@τ)⟩, k)
```

```
frame@τ =
 match τ with
   | init → empty
   | _ → frame@pred(τ) :: out@τ

in@τ =
 match τ with
   | init → empty
   | _ → 🐈( frame@pred(τ) )
```

The CCSA Logic: Reasoning Rules

More complicated with **indexed names**, e.g. $n_R(j_0) \neq$ 🐈$(\mathsf{frame}@\tau)$.
$\Longrightarrow$ use **local formulas** to ensure freshness.

Indices at which n_R is read in 🐈$(\mathsf{frame}@\tau)$:

$$\{j \mid R_1(j) \leq \tau \text{ or } R_2(j) \leq \tau\} = \{j \mid R_1(j) \leq \tau\}$$

```
$\mathsf{out}@\tau$ =
 match $\tau$ with
   | init → empty
   | $T(i) \rightarrow \langle n_T(i), H(\langle \mathsf{in}@\tau, n_T(i)\rangle, k)\rangle$
   | $R_1(j) \rightarrow n_R(j)$
   | $R_2(j) \rightarrow \pi_2(\mathsf{in}@\tau) = H(\langle n_R(j), \pi_1(\mathsf{in}@\tau)\rangle, k)$

$\mathsf{frame}@\tau$ =
 match $\tau$ with
   | init → empty
   | _ → $\mathsf{frame}@\mathsf{pred}(\tau) :: \mathsf{out}@\tau$

$\mathsf{in}@\tau$ =
 match $\tau$ with
   | init → empty
   | _ → 🐈$(\mathsf{frame}@\mathsf{pred}(\tau))$
```

The CCSA Logic: Reasoning Rules

More complicated with **indexed names**, e.g. $n_R(j_0) \neq$ 😈$(\mathsf{frame}@\tau)$.
$\Longrightarrow$ use **local formulas** to ensure freshness.

Indices at which n_R is read in 😈$(\mathsf{frame}@\tau)$:

$$\{j \mid R_1(j) \leq \tau \text{ or } R_2(j) \leq \tau\} = \{j \mid R_1(j) \leq \tau\}$$

Thus, we can take:

$$[\tau < R_1(j_0) \;\Rightarrow\; n_R(j_0) \neq \text{😈}(\mathsf{frame}@\tau)]$$

```
out@τ =
 match τ with
   | init → empty
   | T(i) → ⟨n_T(i), H(⟨in@τ, n_T(i)⟩, k)⟩
   | R₁(j) → n_R(j)
   | R₂(j) → π₂(in@τ) = H(⟨n_R(j), π₁(in@τ)⟩, k)

frame@τ =
 match τ with
   | init → empty
   | _ → frame@pred(τ) :: out@τ

in@τ =
 match τ with
   | init → empty
   | _ → 😈( frame@pred(τ) )
```

The CCSA Logic: Reasoning Rules

Probabilistic reasoning: collision of random samplings

General case: local formula $\phi^{\mathsf{n},i}_{\mathsf{fresh}}(\vec{u})$.

Ensures that $\mathsf{n}(i)$ fresh in $\vec{u}$.

$$\dfrac{\textsc{Indep}}{[\phi^{\mathsf{n},i}_{\mathsf{fresh}}(t,i) \;\Rightarrow\; (t \neq \mathsf{n}(i))]}$$

The CCSA Logic: Reasoning Rules

Probabilistic reasoning: collision of random samplings

General case: local formula $\phi^{\mathsf{n},i}_{\mathsf{fresh}}(\vec{u})$.

Ensures that $\mathsf{n}(i)$ fresh in $\vec{u}$.

$$\textsc{Indep}\quad \frac{}{[\phi^{\mathsf{n},i}_{\mathsf{fresh}}(t,i) \Rightarrow (t \neq \mathsf{n}(i))]}$$

Computing such freshness formulas is **non-trivial**. Indeed:

$$\phi^{\mathsf{n},i}_{\mathsf{fresh}}(f(t)) \iff \text{cell } i \text{ of array } \mathsf{n} \text{ never read in } f(t) \text{ computation}$$

This is **undecidable**.

$\Longrightarrow$ we rely on **approximations**.

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

An obvious **reduction** rule:

$$\frac{\vec{v}_0 \sim \vec{v}_1}{\mathsf{f}(\vec{v}_0) \sim \mathsf{f}(\vec{v}_1)}\ \mathrm{FA} \qquad \text{where } \mathsf{f} \in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{😈} \in \mathcal{F}_{\mathsf{adv}}\}$$

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

An obvious **reduction** rule:

$$\frac{\vec{v}_0 \sim \vec{v}_1}{\mathsf{f}(\vec{v}_0) \sim \mathsf{f}(\vec{v}_1)}\ \mathrm{FA} \qquad \text{where } \mathsf{f} \in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{😈} \in \mathcal{F}_{\mathsf{adv}}\}$$

Proof

Take a model $\mathbb{M}$ and $\mathcal{A}$ against the conclusion.

Take $\mathcal{B}(\vec{v}) := \{\, \mathsf{x} \leftarrow \mathbb{M}_{\mathsf{f}}(\vec{v});\ \textbf{return}\ \mathcal{A}(\mathsf{x})\,\}$.

$\mathcal{B}$ is polynomial-time since $\mathbb{M}_{\mathsf{f}}$ and $\mathcal{A}$ are.

Thus $\mathsf{Adv}(\mathcal{A}) = \mathsf{Adv}(\mathcal{B})$, negligible by hypothesis.

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

An obvious **reduction** rule:

$$\frac{\vec{v}_0 \sim \vec{v}_1}{\mathsf{f}(\vec{v}_0) \sim \mathsf{f}(\vec{v}_1)}\ \mathrm{FA} \qquad \text{where } \mathsf{f} \in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{😈 \in \mathcal{F}_{\mathsf{adv}}\}$$

Proof

Take a model $\mathbb{M}$ and $\mathcal{A}$ against the conclusion.

Take $\mathcal{B}(\vec{v}) := \{\, \mathsf{x} \leftarrow \mathbb{M}_{\mathsf{f}}(\vec{v});\ \textbf{return}\ \mathcal{A}(\mathsf{x})\,\}$.

$\mathcal{B}$ is polynomial-time since $\mathbb{M}_{\mathsf{f}}$ and $\mathcal{A}$ are.

Thus $\mathsf{Adv}(\mathcal{A}) = \mathsf{Adv}(\mathcal{B})$, negligible by hypothesis.

$\Rightarrow$ FA moves a **deterministic computation** in the **top-level adv**.

(or a computation using adversarial randomness)

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Simple **reductions** rules:

$$\frac{\vec{u}_0, \vec{v}_0 \sim \vec{u}_1, \vec{v}_1}{\vec{u}_0, \mathsf{f}(\vec{v}_0) \sim \vec{u}_1, \mathsf{f}(\vec{v}_1)}\ \text{FA} \qquad \text{where } \mathsf{f} \in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{🐈} \in \mathcal{F}_{\mathsf{adv}}\}$$

$$\frac{\begin{array}{c}[\phi^{\mathsf{n},i}_{\mathsf{fresh}}(\vec{u}, i) \mathbin{\tilde{\wedge}} \phi^{\mathsf{m},j}_{\mathsf{fresh}}(\vec{v}, j)] \\ \vec{u} \sim \vec{v}\end{array}}{\vec{u}, \mathsf{n}(i) \sim \vec{v}, \mathsf{m}(j)}\ \textsc{Fresh}$$

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Simple **reductions** rules:

$$\frac{\vec{u}_0, \vec{v}_0 \sim \vec{u}_1, \vec{v}_1}{\vec{u}_0, \mathsf{f}(\vec{v}_0) \sim \vec{u}_1, \mathsf{f}(\vec{v}_1)}\ \text{FA} \qquad \text{where } \mathsf{f} \in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{🐈} \in \mathcal{F}_{\mathsf{adv}}\}$$

$$\frac{[\phi^{\mathsf{n},i}_{\mathsf{fresh}}(\vec{u}, i) \mathbin{\tilde{\wedge}} \phi^{\mathsf{m},j}_{\mathsf{fresh}}(\vec{v}, j)] \quad \vec{u} \sim \vec{v}}{\vec{u}, \mathsf{n}(i) \sim \vec{v}, \mathsf{m}(j)}\ \textsc{Fresh} \qquad \frac{\vec{u}_0, t_0 \sim \vec{u}_1, t_1}{\vec{u}_0, t_0, t_0 \sim \vec{u}_1, t_1, t_1}\ \textsc{Dup}$$

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Simple **reductions** rules:

$$\frac{\vec{u}_0, \vec{v}_0 \sim \vec{u}_1, \vec{v}_1}{\vec{u}_0, \mathsf{f}(\vec{v}_0) \sim \vec{u}_1, \mathsf{f}(\vec{v}_1)}\ \textsc{FA} \qquad \text{where } \mathsf{f} \in \{\mathsf{f} \in \mathcal{F}_{\mathsf{lib}}\} \cup \{\text{😈} \in \mathcal{F}_{\mathsf{adv}}\}$$

$$\frac{\begin{array}{c}[\phi^{\mathsf{n},i}_{\mathsf{fresh}}(\vec{u}, i)\ \tilde{\wedge}\ \phi^{\mathsf{m},j}_{\mathsf{fresh}}(\vec{v}, j)] \\ \vec{u} \sim \vec{v}\end{array}}{\vec{u}, \mathsf{n}(i) \sim \vec{v}, \mathsf{m}(j)}\ \textsc{Fresh} \qquad \frac{\vec{u}_0, t_0 \sim \vec{u}_1, t_1}{\vec{u}_0, t_0, t_0 \sim \vec{u}_1, t_1, t_1}\ \textsc{Dup}$$

⇒ mostly **book-keeping** rules.

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Rules capturing **reduction** to **hardness assumptions**.

$$\text{CPA}\ \frac{[\mathsf{len}(m_0) = \mathsf{len}(m_1)]}{\begin{array}{rl} & \vec{u}, \mathsf{enc}(m_0, \mathsf{k}, \mathsf{r}) \\ \sim & \vec{u}, \mathsf{enc}(m_1, \mathsf{k}, \mathsf{r}) \end{array}}$$

$$\text{PRF}\ \frac{}{\vec{u}, \mathsf{H}(t, \mathsf{k}) \sim \vec{u}, \mathsf{n}_{\mathsf{fresh}}}$$

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Rules capturing **reduction** to **hardness assumptions**.

$$\mathrm{CPA}\ \frac{[\phi_{\mathsf{ekey}}] \qquad [\phi_{\mathsf{rand}}] \qquad [\mathsf{len}(m_0) = \mathsf{len}(m_1)]}{\vec{u}, \mathsf{enc}(m_0, \mathsf{k}, \mathsf{r}) \sim \vec{u}, \mathsf{enc}(m_1, \mathsf{k}, \mathsf{r})}$$

- ϕ_{ekey}: k only used in encryption key position $\mathsf{enc}(\cdot, \mathsf{k}, \cdot)$ with fresh rands.
- ϕ_{rand}: r fresh name.
- $\vec{u}, m_0, m_1$ ptime-computable.

$$\mathrm{PRF}\ \frac{}{\vec{u}, \mathsf{H}(t, \mathsf{k}) \sim \vec{u}, \mathsf{n}_{\mathsf{fresh}}}$$

As for Indep, we have **side-conditions**.

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Rules capturing **reduction** to **hardness assumptions**.

$$\text{CPA}\ \frac{[\phi_{\mathsf{ekey}}] \qquad [\phi_{\mathsf{rand}}] \qquad [\mathsf{len}(m_0) = \mathsf{len}(m_1)]}{\vec{u}, \mathsf{enc}(m_0, \mathsf{k}, \mathsf{r}) \sim \vec{u}, \mathsf{enc}(m_1, \mathsf{k}, \mathsf{r})}$$

- ϕ_{ekey}: k only used in encryption key position $\mathsf{enc}(\cdot, \mathsf{k}, \cdot)$ with fresh rands.
- ϕ_{rand}: r fresh name.
- $\vec{u}, m_0, m_1$ ptime-computable.

$$\text{PRF}\ \frac{[\phi_{\mathsf{hkey}}] \qquad [\phi_{\mathsf{hash}}]}{\vec{u}, \mathsf{H}(t, \mathsf{k}) \sim \vec{u}, \mathsf{n}_{\mathsf{fresh}}}$$

- ϕ_{hkey}: k only used in hash key position $\mathsf{H}(\cdot, \mathsf{k})$.
- ϕ_{hash}: t never hashed by $\mathsf{H}(\cdot, \mathsf{k})$.
- $\vec{u}, t$ ptime-computable.

As for Indep, we have **side-conditions**.

The CCSA Logic: Reasoning Rules

High-level structure

The **induction** rule:

$$\frac{\vec{u}(0) \sim \vec{v}(0) \qquad \tilde{\forall}(N : \mathsf{int}).\ \vec{u}(N) \sim \vec{v}(N) \mathrel{\tilde{\Rightarrow}} \vec{u}(N+1) \sim \vec{v}(N+1)}{\tilde{\forall}(N : \mathsf{int}).\ \vec{u}(N) \sim \vec{v}(N)}$$

The CCSA Logic: Reasoning Rules

High-level structure

The **induction** rule:

$$\frac{\vec{u}(0) \sim \vec{v}(0) \qquad \tilde{\forall}(N : \mathsf{int}).\ \vec{u}(N) \sim \vec{v}(N) \mathrel{\tilde{\Rightarrow}} \vec{u}(N+1) \sim \vec{v}(N+1)}{\tilde{\forall}(N : \mathsf{int}).\ \vec{u}(N) \sim \vec{v}(N)}$$

Only for a **constant** number of steps N.

Same reason as for **hybrid arguments**:

$$\begin{aligned} \vec{u}(0) \sim \cdots \sim \vec{u}(N) &\implies \vec{u}(0) \sim_{f_1(\eta)} \cdots \sim_{f_N(\eta)} \vec{u}(N) \quad ((f_i)_i \text{ negligible}) \\ &\implies \vec{u}(0) \sim_{\sum_{i \leq N} f_i(\eta)} \vec{u}(N) \end{aligned}$$

$\sum_{i \leq N} f_i(\eta)$ may not be negligible if N polynomial in η.

The CCSA Logic: Reasoning Rules

High-level structure

The **induction** rule:

$$\frac{\vec{u}(0) \sim \vec{v}(0) \qquad \tilde{\forall}(N : \mathsf{int}).\ \big(\mathsf{const}(N)\ \tilde{\wedge}\ \vec{u}(N) \sim \vec{v}(N)\big) \mathrel{\tilde{\Rightarrow}} \vec{u}(N+1) \sim \vec{v}(N+1)}{\tilde{\forall}(N : \mathsf{int}).\ \mathsf{const}(N) \mathrel{\tilde{\Rightarrow}} \vec{u}(N) \sim \vec{v}(N)}$$

Only for a **constant** number of steps N.

Same reason as for **hybrid arguments**:

$$\begin{aligned}\vec{u}(0) \sim \cdots \sim \vec{u}(N) &\implies \vec{u}(0) \sim_{f_1(\eta)} \cdots \sim_{f_N(\eta)} \vec{u}(N) \quad ((f_i)_i \text{ negligible})\\ &\implies \vec{u}(0) \sim_{\sum_{i\leq N} f_i(\eta)} \vec{u}(N)\end{aligned}$$

$\sum_{i\leq N} f_i(\eta)$ may not be negligible if N polynomial in η.

Formalizing Cryptographic Proofs

Our **formal framework** must model and capture:

- $\mathcal{P}$: **protocol** ✓
- 😈 $\in \mathcal{C}$: **adversarial model** ✓
- Φ: **security property** ✓
- $\models$: **cryptographic arguments** ✓

We are done with our framework!

The CCSA Logic: Summary

- Logic with a **probabilistic interpretation** of terms:
  protocol execution $\Rightarrow$ terms of the logic.

- **Security predicates** $[\phi]$ and $\vec{u}_0 \sim \vec{u}_1$.
  - **Abstract** predicates: no **probabilities** and **security parameter**.
  - Can express **temporal properties** as formulas $[\phi]$:
    direct quantification on the execution trace (no encoding).

- **Reasoning rules** to capture crypto. arguments:
  - generic math. reasoning
  - game-hopping steps
  - probabilistic arguments
  - crypto. reductions

  The **application conditions** for crypto. and probabilistic rules are the difficult part.

Limitations

Two **limitations** of this CCSA logic:

- **guarantees provided**: parametric vs polynomial security.
- **modularity**: ad hoc rules for a fixed number of crypto. assumptions.

A Concrete Security CCSA Logic

with D. Baelde, C. Fontaine, G. Scerri, T. Vignon

Limitation: Polynomial vs Parametric Security

We reason over a **fixed trace** $\mathcal{T}$ given by $[\![\text{timestamp}]\!]_{\mathbb{M}}$.

This only yields **parametric** security. Informally, $\mathbb{M} \models \Phi$ implies:

$$\forall \mathcal{T}. \forall \mathcal{A}. \ \text{Pr}(\Phi \text{ holds in } \mathcal{T} \text{ against } \mathcal{A}) \text{ is overwhelming in } \eta$$

We expect the stronger **polynomial** security:

$$\forall \mathcal{A}. \ \text{Pr}(\Phi \text{ holds in } \mathcal{T} \text{ chosen by } \mathcal{A}) \text{ is overwhelming in } \eta$$

Limitation: Polynomial vs Parametric Security

How to obtain **polynomial security** using CCSA [Bae+24, to appear]:

- Change the **execution model**.
  E.g. frame@N where (N : int) instead of frame@τ.
- **Difficulty**: previous induction rule requires a constant number of steps.
  because $\sum_{i \leq P(\eta)} f_i(\eta)$ is not always negligible,
  even if $f_i(\eta)$ negligible $\forall i$ and $P(\eta)$ polynomial.
- Solution: move to a **concrete security** setting.
  - **concrete security predicates** $[\phi]_\epsilon$ and $\vec{u}_0 \sim_\epsilon \vec{u}_1$.
  - reasoning rules with **explicit bounds**.
  - support **general induction**:
    user must prove a uniform bound on all f_i's.
- For now, theoretical work (implementation in Squirrel is WIP).

From Hardness Assumptions to Logical Rules

with D. Baelde, J. Sauvage

Hardness Assumption: Example

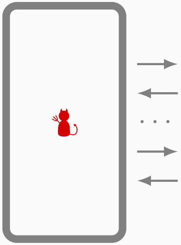
A **cryptographic hash** function $\mathsf{H}(\overset{\text{message}}{m}, \overset{\text{key}}{\mathsf{key}})$.

Unforgeability: cannot produce valid hashes without knowing key.

Hardness Assumption: Example

$$\text{A \textbf{cryptographic hash} function } \mathsf{H}(\underbrace{m}_{\text{message}}, \underbrace{\mathsf{key}}_{\text{key}}).$$

Unforgeability: cannot produce valid hashes without knowing key.



```
Init: key $\xleftarrow{\$}$;

$\mathcal{O}_{\text{hash}}(m_0)$ :=
    $\mathcal{L} \leftarrow m_0 :: \mathcal{L}$
    return $\mathsf{H}(m_0, \mathsf{key})$

$\mathcal{O}_{\text{challenge}}(m, s)$ :=
    return $\begin{cases} m \notin \mathcal{L} \text{ and } s = \mathsf{H}(m, \mathsf{key}) & \text{(left game)} \\ \text{false} & \text{(right game)} \end{cases}$
```

Hardness Assumption: Example

Example

$$🐈‍⬛\Big(\mathsf{H}(0,\mathsf{k}),\mathsf{H}(1,\mathsf{k})\Big) = \mathsf{H}(m,\mathsf{k}) \quad \Rightarrow \quad m = 0 \ \vee \ m = 1$$

Proof by reduction

Build an adversary 😈 against UNFORGEABILITY (UF):

- compute $h_0 \leftarrow \mathcal{O}_{\mathsf{hash}}(0)$ and $h_1 \leftarrow \mathcal{O}_{\mathsf{hash}}(1)$;
- black-box call: $s \leftarrow 🐈‍⬛(h_0, h_1)$;
- compute m;
- return $\mathcal{O}_{\mathsf{challenge}}(m, s)$.

$\mathsf{Adv}_{\mathsf{UF}}(😈) = \mathsf{Adv}(🐈‍⬛)$ $\qquad$ $🐈‍⬛ \in \mathrm{PPTM}$ implies $😈 \in \mathrm{PPTM}$

Hardness Assumption: Example

Example

$$\text{🐈‍⬛}\Big(\mathsf{H}(0,\mathsf{k}),\mathsf{H}(1,\mathsf{k})\Big) = \mathsf{H}(m,\mathsf{k}) \quad \Rightarrow \quad m = 0 \;\vee\; m = 1$$

Proof by reduction

Build an adversary 😈 against Unforgeability (UF):

- compute $h_0 \leftarrow \mathcal{O}_{\mathsf{hash}}(0)$ and $h_1 \leftarrow \mathcal{O}_{\mathsf{hash}}(1)$;
- black-box call: $s \leftarrow \text{🐈‍⬛}(h_0, h_1)$;
- compute m;
- return $\mathcal{O}_{\mathsf{challenge}}(m, s)$.

$$\mathsf{Adv}_{\mathsf{UF}}(\text{😈}) = \mathsf{Adv}(\text{🐈‍⬛}) \qquad \text{🐈‍⬛} \in \mathrm{PPTM} \text{ implies } \text{😈} \in \mathrm{PPTM}$$

Remark: rule valid only if m computable by the adversary.

From Hardness Assumptions to Logical Rules

Until recently:

- Squirrel supported a limited set of hardness assumptions (symmetric/asymmetric encryption, signature, hash, DH, . . .)
- Built-in tactics for each such assumptions:

  hardness assumption (imperative, stateful programs)

  $\Downarrow$

  reasoning rules (pure, logic)

- Adding rules for new hardness assumptions is: **tedious**, **error-prone**, and **not in user-space** (Ocaml code).

From Hardness Assumptions to Logical Rules

Systematic cryptographic reductions: allows to translate hardness assumptions into cryptographic rules.

Inputs:

- an (imperative, stateful) **hardness assumption** $\mathcal{G}_0 \approx \mathcal{G}_1$.
- an **indistinguishability property**, e.g. $u_0 \sim u_1$ to prove, i.e.:

$$\forall \text{😈}.\ \left| \Pr(\text{😈}(\llbracket u_0 \rrbracket)) - \Pr(\text{😈}(\llbracket u_1 \rrbracket)) \right| \leq \mathsf{negl}(\eta)$$

Goal: synthesize $\mathcal{S}$ poly-time such that $\begin{cases} \mathcal{S}^{\mathcal{G}_0}() = \llbracket u_0 \rrbracket \\ \text{and } \mathcal{S}^{\mathcal{G}_1}() = \llbracket u_1 \rrbracket \end{cases}$

Thus, for any 😈:

$$\mathsf{Adv}_{u_0 \sim u_1}(\text{😈}) = \mathsf{Adv}_{\mathcal{G}_0 \approx \mathcal{G}_1}(\text{😈} \circ \mathcal{S}) \leq \mathsf{negl}(\eta)$$

From Hardness Assumptions to Logical Rules

- **General framework** to add new hardness assumptions.
- **Proof system** to establish the existence of $\mathcal{S}$.
- **Fully automated implementation** (heuristic based $\Rightarrow$ incomplete)

Bi-Deduction

Take an **hardness assumption** $\mathcal{G}_0 \approx \mathcal{G}_1$.

Bi-Terms

The **bi-terms** $u_{\#} = \#(u_0; u_1)$ represent a pair of left/right scenarios.

Factorize common behavior, e.g. $f(v, \#(u_0; u_1)) = \#(f(v, u_0); f(v, u_1))$

Bi-Deduction

Take an **hardness assumption** $\mathcal{G}_0 \approx \mathcal{G}_1$.

Bi-Terms

The **bi-terms** $u_\# = \#(u_0; u_1)$ represent a pair of left/right scenarios.

Factorize common behavior, e.g. $f(v, \#(u_0; u_1)) = \#(f(v, u_0); f(v, u_1))$

Bi-deduction

New predicate $u_\# \triangleright_{\mathcal{G}_0 \approx \mathcal{G}_1} v_\#$ which means:

$$\exists \mathcal{S} \in \mathrm{PPTM}.\ \begin{cases} \mathcal{S}^{\mathcal{G}_0}(\llbracket u_0 \rrbracket) = \llbracket v_0 \rrbracket \\ \text{and } \mathcal{S}^{\mathcal{G}_1}(\llbracket u_1 \rrbracket) = \llbracket v_1 \rrbracket \end{cases}$$

Bi-Deduction

Take an **hardness assumption** $\mathcal{G}_0 \approx \mathcal{G}_1$.

Bi-Terms

The **bi-terms** $u_\# = \#(u_0; u_1)$ represent a pair of left/right scenarios.

Factorize common behavior, e.g. $f(v, \#(u_0; u_1)) = \#(f(v, u_0); f(v, u_1))$

Bi-deduction

New predicate $u_\# \triangleright_{\mathcal{G}_0 \approx \mathcal{G}_1} v_\#$ which means:

$$\exists \mathcal{S} \in \mathrm{PPTM}. \begin{cases} \mathcal{S}^{\mathcal{G}_0}(\llbracket u_0 \rrbracket) = \llbracket v_0 \rrbracket \\ \text{and } \mathcal{S}^{\mathcal{G}_1}(\llbracket u_1 \rrbracket) = \llbracket v_1 \rrbracket \end{cases}$$

Inference Rule

$$\frac{\emptyset \triangleright_{\mathcal{G}_0 \approx \mathcal{G}_1} \#(u_0; u_1)}{u_0 \sim u_1} \textsc{Bi-Deduce}$$

Bi-Deduction: Rules

A few simple **bi-deduction rules**:

- **Transitivity**

$$\frac{\vec{u}_\# \rhd \vec{v}_\# \qquad \vec{u}_\#, \vec{v}_\# \rhd \vec{w}_\#}{\vec{u}_\# \rhd \vec{v}_\#, \vec{w}_\#}$$

```
$\underline{\mathcal{S}(\vec{u})}$ := $\vec{v} \leftarrow \mathcal{S}_1(\vec{u})$
         $\vec{w} \leftarrow \mathcal{S}_2(\vec{u}, \vec{v})$
         return $(\vec{v}, \vec{w})$
```

Bi-Deduction: Rules

A few simple **bi-deduction rules**:

- **Transitivity**

$$\frac{\vec{u}_\# \rhd \vec{v}_\# \qquad \vec{u}_\#, \vec{v}_\# \rhd \vec{w}_\#}{\vec{u}_\# \rhd \vec{v}_\#, \vec{w}_\#}$$

```
$\underline{\mathcal{S}}(\vec{u}) :=$  $\vec{v} \leftarrow \mathcal{S}_1(\vec{u})$
          $\vec{w} \leftarrow \mathcal{S}_2(\vec{u}, \vec{v})$
          return $(\vec{v}, \vec{w})$
```

- **Function application** (where $\mathsf{f} \in \mathcal{F}_{\mathsf{lib}} \cup \mathcal{F}_{\mathsf{adv}}$)

$$\frac{\vec{u}_\# \rhd \vec{v}_\#}{\vec{u}_\# \rhd \mathsf{f}(\vec{v}_\#)}$$

```
$\underline{\mathcal{S}}(\vec{u}) :=$  $\vec{v} \leftarrow \mathcal{S}_1(\vec{u})$
          $x \leftarrow \mathbb{M}_{\mathsf{f}}(\vec{v})$
          return $x$
```

Bi-Deduction: Rules

Bi-deduction rules handling **randomness**:

ORACLE

$$\frac{\vec{u}_\# \triangleright v_\#}{\vec{u}_\# \triangleright \mathsf{H}(v_\#, \mathsf{k})}$$

$$\underline{\mathcal{S}}(\vec{u}) := \begin{array}{l} \vec{v} \leftarrow \mathcal{S}_1(\vec{u}) \\ x \overset{\$}{\leftarrow} \mathcal{O}_{\mathsf{hash}}(\vec{v}) \\ \textbf{return } x \end{array}$$

NAME

$$\frac{\vec{u}_\# \triangleright v_\#}{\vec{u}_\# \triangleright \mathsf{n}(v_\#)}$$

$$\underline{\mathcal{S}}(\vec{u}) := \begin{array}{l} v \leftarrow \mathcal{S}_1(\vec{u}) \\ x \overset{\$}{\leftarrow} \mathbb{M}_{\mathsf{n_f}}(v, \rho_h) \\ \textbf{return } x \end{array}$$

Bi-Deduction: Rules

Bi-deduction rules handling **randomness**:

Oracle
$$\frac{\vec{u}_\# \rhd v_\#}{\vec{u}_\# \rhd \mathsf{H}(v_\#, \mathsf{k})}$$

```
$\underline{\mathcal{S}}(\vec{u}) :=$ $\vec{v} \leftarrow \mathcal{S}_1(\vec{u})$
         $x \xleftarrow{\$} \mathcal{O}_{\mathsf{hash}}(\vec{v})$
         return $x$
```

Name
$$\frac{\vec{u}_\# \rhd v_\#}{\vec{u}_\# \rhd \mathsf{n}(v_\#)}$$

```
$\underline{\mathcal{S}}(\vec{u}) :=$ $v \leftarrow \mathcal{S}_1(\vec{u})$
         $x \xleftarrow{\$} \mathbb{M}_{\mathsf{n_f}}(v, \rho_h)$
         return $x$
```

Problem: the Name rule allow $\mathcal{S}$ to read k!

Bi-Deduction: Constraints

- **Problem:** $\mathcal{S}$ should not **access the game secret keys**.
- **Solution:** track **randomness usage** using logical **constraints**.
  E.g. ensures that $\mathcal{S}$ does not directly use key.
- Annotated bi-deduction predicate:

$$\textsc{Oracle}\ \frac{\vdash \vec{u}_{\#} \triangleright v_{\#}}{(k : \mathsf{T}^{\mathsf{key}}_{\mathsf{G}}) \vdash \vec{u}_{\#} \triangleright \mathsf{H}(v_{\#}, k)} \qquad \textsc{Name}\ \frac{}{(n : \mathsf{T}_{\mathcal{S}}) \vdash \vec{u}_{\#} \triangleright n}$$

Bi-Deduction: Constraints

Eventually, check that the **constraints** are **valid**:

$$\frac{\mathcal{C} \vdash \emptyset \rhd \#(u_0; u_1) \qquad \models [\mathsf{Valid}(\mathcal{C})]}{u_0 \sim u_1}\ \textsc{Bi-Deduce}$$

Example:

$$\not\models [\mathsf{Valid}((\mathsf{k} : \mathsf{T}_{\mathsf{G}}^{\mathsf{key}}), (\mathsf{k} : \mathsf{T}_{\mathcal{S}}))]$$

Bi-Deduction: Constraints

Eventually, check that the **constraints** are **valid**:

$$\frac{\mathcal{C} \vdash \emptyset \rhd \#(u_0; u_1) \qquad \models [\mathsf{Valid}(\mathcal{C})]}{u_0 \sim u_1}\ \textsc{Bi-Deduce}$$

Example:

$$\not\models [\mathsf{Valid}((\mathsf{k} : \mathsf{T}_{\mathsf{G}}^{\mathsf{key}}), (\mathsf{k} : \mathsf{T}_{\mathcal{S}}))]$$

Some **additional difficulties**:

- We need to handle **indexed names** and **conditions**:

$$(\mathsf{n}, i, \phi : \mathsf{T})$$

- Some weird constraints must be avoided, e.g.:

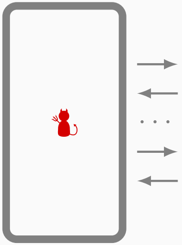
$$(\mathsf{n}, \mathsf{n} = 0, \mathsf{T}_{\mathcal{S}}) \quad \wedge \quad (\mathsf{n}, \mathsf{n} \neq 0, \mathsf{T}_{\mathcal{G}})$$

Bi-Deduction: Statefulness

We also need to account for $\mathcal{G}$'s **statefulness**.

Bi-Deduction: Statefulness

We also need to account for $\mathcal{G}$'s **statefulness**.



```
Init: $\mathsf{key} \xleftarrow{\$}$;

$\mathcal{O}_{\mathsf{hash}}(m_0)$ :=
  $\mathcal{L} \leftarrow m_0 :: \mathcal{L}$
  return $\mathsf{H}(m_0, \mathsf{key})$

$\mathcal{O}_{\mathsf{challenge}}(m, s)$ :=
  return $\begin{cases} m \notin \mathcal{L} \text{ and } s = \mathsf{H}(m, \mathsf{key}) & \text{(left game)} \\ \mathsf{false} & \text{(right game)} \end{cases}$
```

Bi-Deduction: Statefulness

We **track the state** of $\mathcal{G}$:

- Add **Hoare pre- and post- conditions**:

$$(\phi, \psi) \vdash u_{\#} \triangleright v_{\#}$$

- **Semantics**:

$$\exists \mathcal{S} \in \mathrm{PPTM}.\ \forall \mu \models \phi. \quad (\!|\mathcal{S}|\!)_{\mu}^{\mathcal{G}_i}(u_i) = (\mu', [\![v_i]\!]) \qquad (\forall i \in \{0,1\})$$
$$\text{where } \mu' \models \psi$$

Bi-Deduction: Statefulness

We **track the state** of $\mathcal{G}$:

- Add **Hoare pre- and post- conditions**:

$$(\phi, \psi) \vdash u_{\#} \triangleright v_{\#}$$

- **Semantics**:

$$\exists \mathcal{S} \in \mathrm{PPTM}.\ \forall \mu \models \phi. \quad (\!|\mathcal{S}|\!)^{\mathcal{G}_i}_{\mu}(u_i) = (\mu', [\![v_i]\!]) \quad (\forall i \in \{0,1\})$$
$$\text{where } \mu' \models \psi$$

- Modified **proof-system**:

$$\frac{(\phi, \chi) \vdash \vec{u}_{\#} \triangleright \vec{v}_{\#} \qquad (\chi, \psi) \vdash \vec{u}_{\#}, \vec{v}_{\#} \triangleright \vec{w}_{\#}}{(\phi, \psi) \vdash \vec{u}_{\#} \triangleright \vec{v}_{\#}, \vec{w}_{\#}}\ \textsc{Trans}$$

Conclusion: From Hardness Assumptions to Logical Rules

- **Framework** to add new hardness assumptions using **bi-deduction**.
- **Proof system for bi-deduction**.
  - Correct randomness usage using logical **constraints**.
    E.g. ensures that $\mathcal{S}$ does not directly use k.
  - Tracking the state of $\mathcal{G}$: **Hoare pre- and post-conditions**.
    E.g. track the set of hashed messages $\mathcal{L}$.
  - Soundness: existence of a suitable **probabilistic coupling**.
- **Implementation: fully automated** (heuristic based $\Rightarrow$ incomplete).
  Approximate $\mathcal{G}$ state + randomness constraints (discharged to Squirrel).

Conclusion

Conclusion

- The **CCSA logic** behind Squirrel.
  - Modeling protocols as pure terms.
  - Reasoning rules to capture crypto. arguments.
- **Concrete security** variant of the logic.
- **Framework** to add new hardness assumptions using **bi-deduction**.
- Project web-page:

  `https://squirrel-prover.github.io/`

Conclusion

- The **CCSA logic** behind Squirrel.
  - Modeling protocols as pure terms.
  - Reasoning rules to capture crypto. arguments.
- **Concrete security** variant of the logic.
- **Framework** to add new hardness assumptions using **bi-deduction**.
- Project web-page:

  `https://squirrel-prover.github.io/`

Thank you for your attention

References

[Adr+15] David Adrian et al. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *CCS*. ACM, 2015, pp. 5–17.

[Bae+24] David Baelde et al. “A Probablistic Logic for Concrete Security”. In: *IEEE CSF’24, to appear*. 2024.

[Bar+23] Manuel Barbosa et al. “Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and Dilithium”. In: *CRYPTO (5)*. Vol. 14085. Lecture Notes in Computer Science. Springer, 2023, pp. 358–389.

[Bha+14] Karthikeyan Bhargavan et al. “Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS”. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2014, pp. 98–113.

[BR94] Mihir Bellare and Phillip Rogaway. “Optimal Asymmetric Encryption”. In: *EUROCRYPT*. Vol. 950. Lecture Notes in Computer Science. Springer, 1994, pp. 92–111.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. “A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model”. In: *EUROCRYPT (3)*. Vol. 10822. Lecture Notes in Computer Science. Springer, 2018, pp. 552–586.

[Lyu12] Vadim Lyubashevsky. “Lattice Signatures without Trapdoors”. In: *EUROCRYPT*. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 738–755.

[Sho02] Victor Shoup. “OAEP Reconsidered”. In: *J. Cryptol.* 15.4 (2002), pp. 223–249.

The CCSA Logic: Reasoning Rules

Cryptographic reasoning

Reduction to **hardness assumptions** using specific rules.

E.g. for PRF:

$$\mathrm{PRF}\quad \frac{[\phi^{\mathsf{k}}_{\mathsf{hkey}}(\vec{u}, t)]}{\vec{u}, \mathsf{H}(t, \mathsf{k}) \sim \vec{u}, \text{if } \phi^{\mathsf{k},t}_{\mathsf{hash}}(\vec{u}, t) \begin{array}{l}\text{then } \mathsf{n}_{\mathsf{fresh}} \\ \text{else } \mathsf{H}(t, \mathsf{k})\end{array}}$$

- $\phi^{\mathsf{k}}_{\mathsf{hkey}}(\vec{w})$: k only used in hash key position $\mathsf{H}(\cdot, \mathsf{k})$ in $\vec{w}$.
- $\phi^{\mathsf{k},t}_{\mathsf{hash}}(\vec{w})$: t was never hashed by $\mathsf{H}(\cdot, \mathsf{k})$ in $\vec{w}$.

$$\left(\phi^{\mathsf{k},t}_{\mathsf{hash}}(\vec{w}) \wedge m \text{ hashed by } \mathsf{k} \text{ in } \vec{w}\right) \Rightarrow m \neq t$$

The CCSA Logic: Reasoning Rules

Example: messages hashed by k in 🐈‍⬛$(\mathsf{frame@}\tau_0)$:

$$\{\langle \mathsf{in@T(i)}, \mathsf{n_T(i)} \rangle \mid \mathsf{T(i)} \leq \tau_0\}$$
$$\cup\ \{\langle \mathsf{n_R(j)}, \pi_1(\mathsf{in@R_2(j)}) \rangle \mid \mathsf{R_2(j)} \leq \tau_0\}$$

```
out@$\tau$ =
 match $\tau$ with
   | init → empty
   | T(i) → $\langle \mathsf{n_T(i)}, \mathsf{H}(\langle \mathsf{in@}\tau, \mathsf{n_T(i)} \rangle, \mathsf{k}) \rangle$
   | R$_1$(j) → $\mathsf{n_R(j)}$
   | R$_2$(j) → $\pi_2(\mathsf{in@}\tau) = \mathsf{H}(\langle \mathsf{n_R(j)}, \pi_1(\mathsf{in@}\tau) \rangle, \mathsf{k})$
```

```
frame@$\tau$ =
 match $\tau$ with
   | init → empty
   | _ → frame@pred($\tau$) :: out@$\tau$

in@$\tau$ =
 match $\tau$ with
   | init → empty
   | _ → 🐈‍⬛( frame@pred($\tau$) )
```

The CCSA Logic: Reasoning Rules

Example: messages hashed by k in 🐈‍⬛(frame@τ_0):

$$\{\langle \mathsf{in@T(i)}, \mathsf{n_T(i)}\rangle \mid \mathsf{T(i)} \leq \tau_0\}$$
$$\cup\ \{\langle \mathsf{n_R(j)}, \pi_1(\mathsf{in@R_2(j)})\rangle \mid \mathsf{R_2(j)} \leq \tau_0\}$$

Thus, we can take:

$$\phi^{\mathsf{k},t}_{\mathsf{hash}}(\text{🐈‍⬛}(\mathsf{frame@}\tau_0)) \overset{\mathsf{def}}{=} \quad \forall \mathtt{i}.\ \mathsf{T(i)} \leq \tau_0 \ \Rightarrow t \neq \langle \mathsf{in@T(i)}, \mathsf{n_T(i)}\rangle$$
$$\wedge\ \forall \mathtt{j}.\ \mathsf{R_2(j)} \leq \tau_0 \Rightarrow t \neq \langle \mathsf{n_R(j)}, \pi_1(\mathsf{in@R_2(j)})\rangle$$

```
out@τ =
 match τ with
   | init → empty
   | T(i) → ⟨n_T(i), H(⟨in@τ, n_T(i)⟩, k)⟩
   | R₁(j) → n_R(j)
   | R₂(j) → π₂(in@τ) = H(⟨n_R(j), π₁(in@τ)⟩, k)
```

```
frame@τ =
 match τ with
   | init → empty
   | _ → frame@pred(τ) :: out@τ

in@τ =
 match τ with
   | init → empty
   | _ → 🐈‍⬛( frame@pred(τ) )
```

The CCSA Logic: Reasoning Rules

Example: weak privacy for Hash-Lock.

$$\mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}$$

where $t \overset{\text{def}}{=} \langle \mathsf{in}@\mathsf{T}(\mathtt{i}_0)\, ,\, \mathsf{n}_{\mathsf{T}}(\mathtt{i}_0)\, \rangle$.

Since $\mathsf{in}@\mathsf{T}(\mathtt{i}_0) =$ 😈$(\mathsf{frame}@\mathsf{T}(\mathtt{i}_0))$, same scenario as previous slide!

The CCSA Logic: Reasoning Rules

Example: weak privacy for Hash-Lock.

$$\mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}$$

where $t \overset{\mathsf{def}}{=} \langle \mathsf{in}@\mathsf{T}(\mathtt{i}_0)\, ,\, \mathsf{n}_{\mathsf{T}}(\mathtt{i}_0)\ \rangle$.

Since $\mathsf{in}@\mathsf{T}(\mathtt{i}_0) =$ 😈$(\mathsf{frame}@\mathsf{T}(\mathtt{i}_0))$, same scenario as previous slide!

Thus, using PRF+Rewrite:

$$\frac{\left[\begin{array}{l} \quad \forall \mathtt{i}.\ \mathsf{T}(\mathtt{i}) < \mathsf{T}(\mathtt{i}_0) \ \Rightarrow t \neq \langle \mathsf{in}@\mathsf{T}(\mathtt{i})\, ,\, \mathsf{n}_{\mathsf{T}}(\mathtt{i})\ \rangle \\ \wedge\ \forall \mathtt{j}.\ \mathsf{R}_2(\mathtt{j}) < \mathsf{T}(\mathtt{i}_0) \Rightarrow t \neq \langle \mathsf{n}_{\mathsf{R}}(\mathtt{j})\, ,\, \pi_1(\mathsf{in}@\mathsf{R}_2(\mathtt{j}))\ \rangle \end{array}\right]}{\mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}}$$

The CCSA Logic: Reasoning Rules

Example: weak privacy for Hash-Lock.

$$\mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}$$

where $t \overset{\mathsf{def}}{=} \langle \mathsf{in}@\mathsf{T}(\mathtt{i}_0)\,,\ \mathsf{n}_{\mathsf{T}}(\mathtt{i}_0) \rangle$.

Since $\mathsf{in}@\mathsf{T}(\mathtt{i}_0) =$ 😈$(\mathsf{frame}@\mathsf{T}(\mathtt{i}_0))$, same scenario as previous slide!

Thus, using PRF+Rewrite:

$$\frac{\left[\begin{array}{l} \quad \forall \mathtt{i}.\ \mathsf{T}(\mathtt{i}) < \mathsf{T}(\mathtt{i}_0) \ \Rightarrow t \neq \langle \mathsf{in}@\mathsf{T}(\mathtt{i})\,,\ \mathsf{n}_{\mathsf{T}}(\mathtt{i}) \rangle \\ \wedge\ \forall \mathtt{j}.\ \mathsf{R}_2(\mathtt{j}) < \mathsf{T}(\mathtt{i}_0) \Rightarrow t \neq \langle \mathsf{n}_{\mathsf{R}}(\mathtt{j})\,,\ \pi_1(\mathsf{in}@\mathsf{R}_2(\mathtt{j})) \rangle \end{array}\right]}{\mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathsf{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}}$$

The CCSA Logic: Reasoning Rules

Example: weak privacy for Hash-Lock.

$$\mathsf{frame}@\mathsf{pred}(\mathtt{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathtt{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}$$

where $t \overset{\mathsf{def}}{=} \langle \mathsf{in}@\mathtt{T}(\mathtt{i}_0)\,,\ \mathsf{n}_{\mathtt{T}}(\mathtt{i}_0) \rangle$.

Since $\mathsf{in}@\mathtt{T}(\mathtt{i}_0) =$ 🐈$(\mathsf{frame}@\mathtt{T}(\mathtt{i}_0))$, same scenario as previous slide!

Thus, using PRF+Rewrite:

$$\frac{\left[\begin{array}{l} \quad\ \forall \mathtt{i}.\ \mathtt{T}(\mathtt{i}) < \mathtt{T}(\mathtt{i}_0) \ \Rightarrow t \neq \langle \mathsf{in}@\mathtt{T}(\mathtt{i})\,,\ \mathsf{n}_{\mathtt{T}}(\mathtt{i}) \rangle \\ \wedge\ \ \forall \mathtt{j}.\ \mathtt{R}_2(\mathtt{j}) < \mathtt{T}(\mathtt{i}_0) \Rightarrow t \neq \langle \mathsf{n}_{\mathtt{R}}(\mathtt{j})\,,\ \pi_1(\mathsf{in}@\mathtt{R}_2(\mathtt{j})) \rangle \end{array}\right]}{\mathsf{frame}@\mathsf{pred}(\mathtt{T}(\mathtt{i}_0)), \mathsf{H}(t, \mathsf{k}) \sim \mathsf{frame}@\mathsf{pred}(\mathtt{T}(\mathtt{i}_0)), \mathsf{n}_{\mathsf{fresh}}}$$

Concludes using generic maths. reasoning + twice Indep to show:

$$\mathtt{T}(\mathtt{i}) < \mathtt{T}(\mathtt{i}_0) \Rightarrow \mathsf{n}_{\mathtt{T}}(\mathtt{i}_0) \neq \mathsf{n}_{\mathtt{T}}(\mathtt{i})$$

$$\mathtt{R}_2(\mathtt{j}) < \mathtt{T}(\mathtt{i}_0) \Rightarrow \mathsf{n}_{\mathtt{T}}(\mathtt{i}_0) \neq \pi_1(\mathsf{in}@\mathtt{R}_2(\mathtt{j}))$$