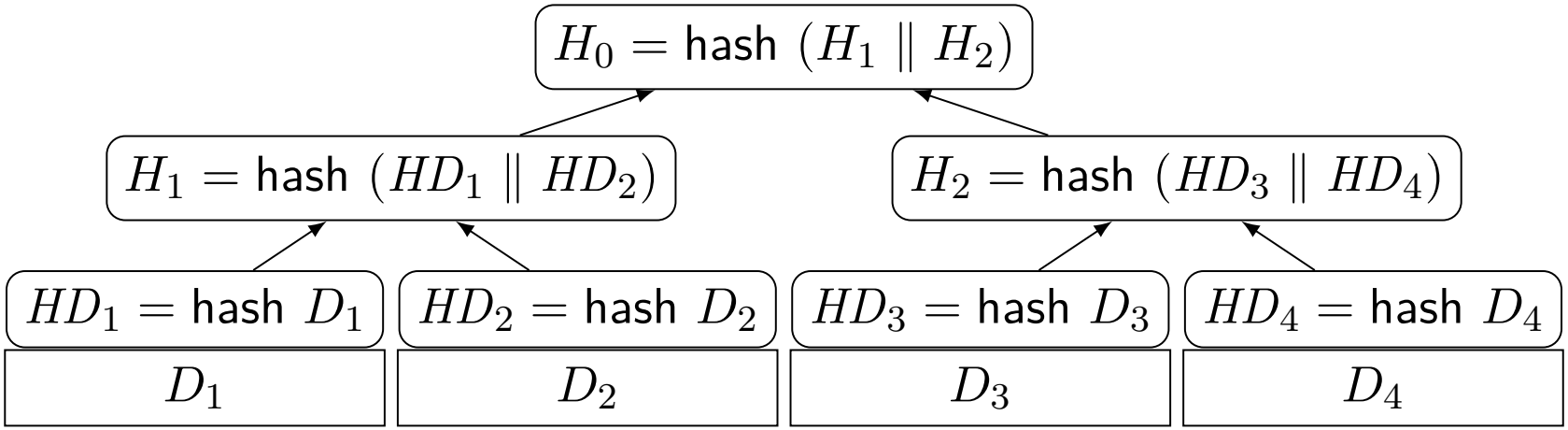
Generic Authenticated Data Structures, Formally

Matthias Brun Dmitriy Traytel

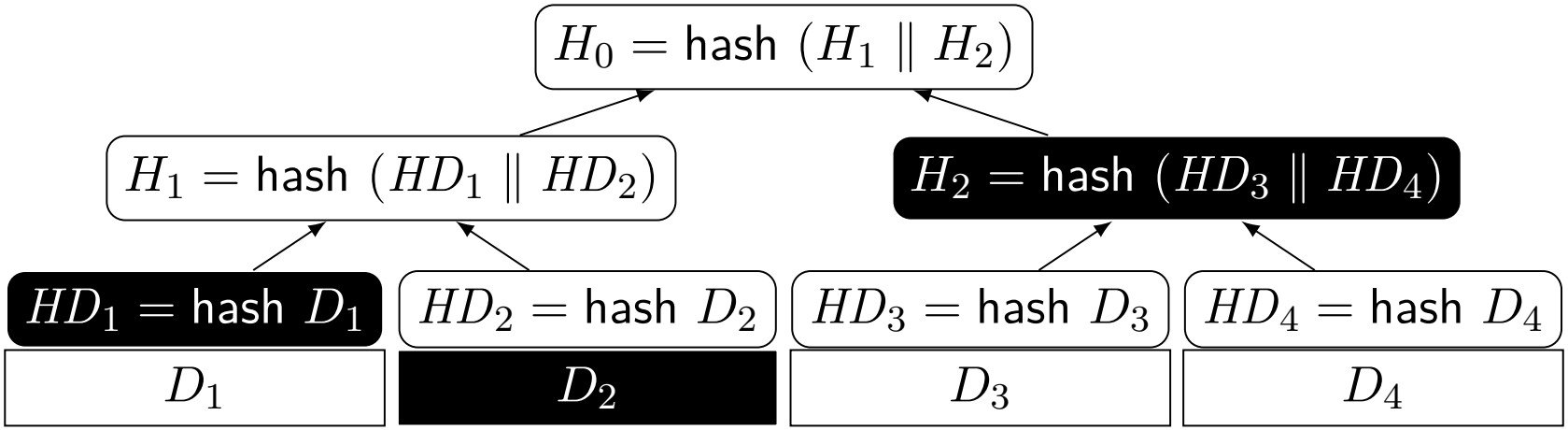
ETH zürich

2019-09-11, ITP'19

Merkle Trees



Merkle Trees



Can Merkle Trees be generalized?

Can Merkle Trees be generalized?

Miller et al. at POPL 2014: Yes $\longrightarrow$ $\lambda\bullet$

Authenticated Data Structures, Generically

Andrew Miller, Michael Hicks, Jonathan Katz, and Elaine Shi
University of Maryland, College Park, USA

Abstract

An authenticated data structure (ADS) is a data structure whose operations can be carried out by an untrusted *prover*, the results of which a *verifier* can efficiently check as authentic. This is done by having the prover produce a compact proof that the verifier can check along with each operation's result. ADSs thus support outsourcing data maintenance and processing tasks to untrusted servers without loss of integrity. Past work on ADSs has focused on particular data structures (or limited classes of data structures), one at a time, often with support only for particular operations.

This paper presents a generic method, using a simple extension to a ML-like functional programming language we call $\lambda\bullet$ (lambda-auth), with which one can program authenticated operations over any data structure defined by standard type constructors, including recursive types, sums, and products. The programmer writes the data structure largely as usual and it is compiled to code to be run by the prover and verifier. Using a formalization of $\lambda\bullet$ we prove that all well-typed $\lambda\bullet$ programs result in code that is secure under the standard cryptographic assumption of collision-resistant hash functions. We have implemented $\lambda\bullet$ as an extension to the OCaml compiler, and have used it to produce authenticated versions of many interesting data structures including binary search

Such a scenario can be supported using *authenticated data structures* (ADS) [5, 23, 30]. ADS computations involve two roles, the *prover* and the *verifier*. The mirror plays the role of the prover, storing the data of interest and answering queries about it. The client plays the role of the verifier, posing queries to the prover and verifying that the returned results are authentic. At any point in time, the verifier holds only a short *digest* that can be viewed as summarizing the current contents of the data; an authentic copy of the digest is provided by the data owner. When the verifier sends the prover a query, the prover computes the result and returns it along with a *proof* that the returned result is correct; both the proof and the time to produce it are linear in the time to compute the query result. The verifier can attempt to verify the proof (in time linear in the size of the proof) using its current digest, and will accept the returned result only if the proof verifies. If the verifier is also the data provider, the verifier may also update its data stored at the prover; in this case, the result is an updated digest and the proof shows that this updated digest was computed correctly. ADS computations have two properties. *Correctness* implies that when both parties execute the protocol correctly, the proofs given by the prover verify correctly and the verifier always receives the correct result. *Security*[1] implies that a computationally bounded, malicious

Miller et al.'s results

$\lambda\bullet$, a small, purely functional programming language

Allows easy specification of authenticated data structures

Correctness and security proofs for specified ADS

Miller et al.'s results

$\lambda\bullet$, a small, purely functional programming language

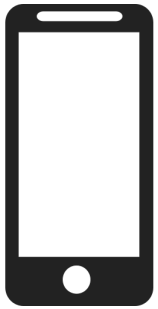
Allows easy specification of authenticated data structures

Correctness and security proofs for specified ADS

Can we confirm these results in Isabelle/HOL?

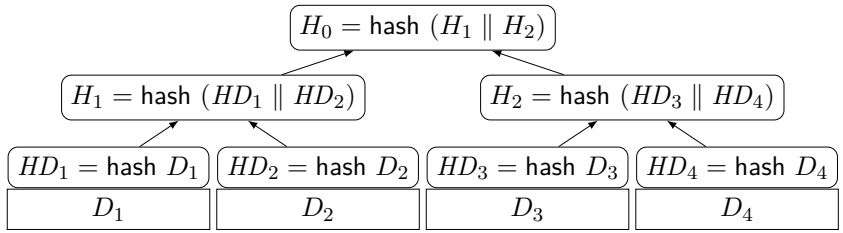
Merkle tree evaluation

Verifier



Prover

$H_0 = \mathsf{hash}\ (H_1 \parallel H_2)$

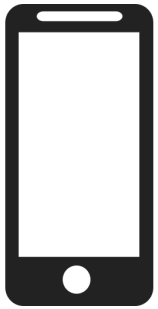


Merkle tree evaluation

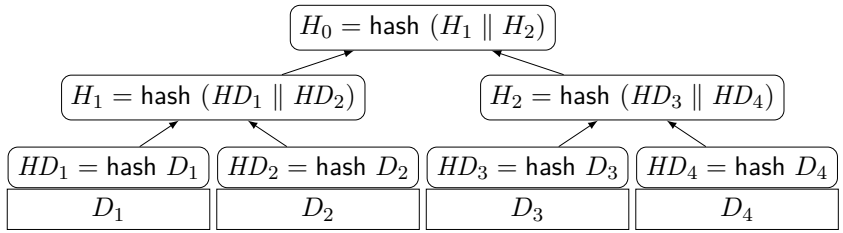
Verifier

fetch [L, R]

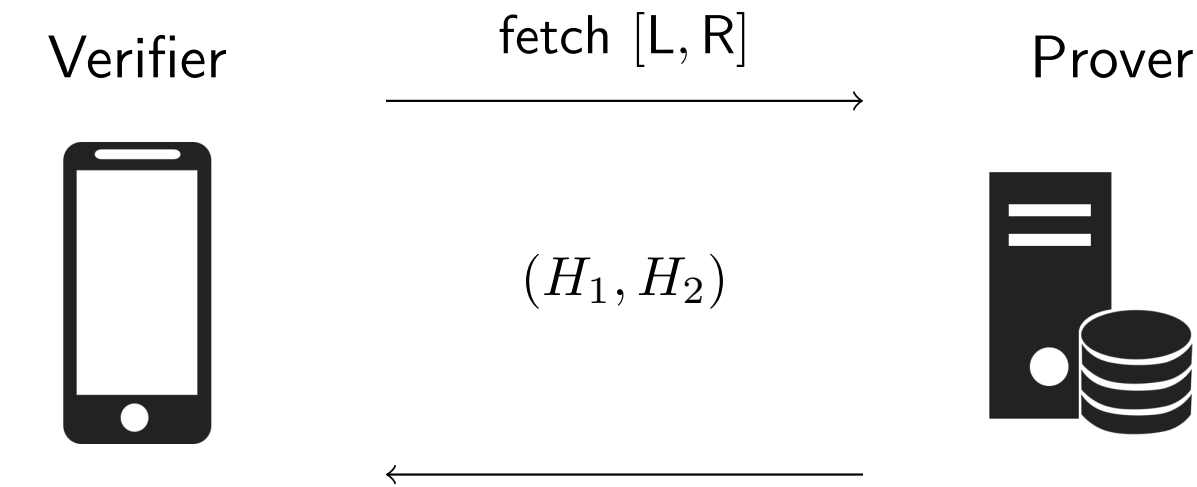
Prover



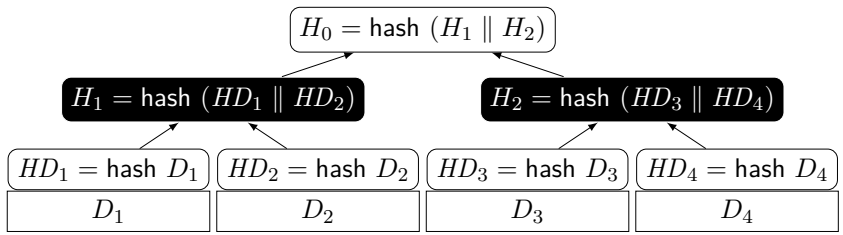
$H_0 = \mathsf{hash}\ (H_1 \parallel H_2)$



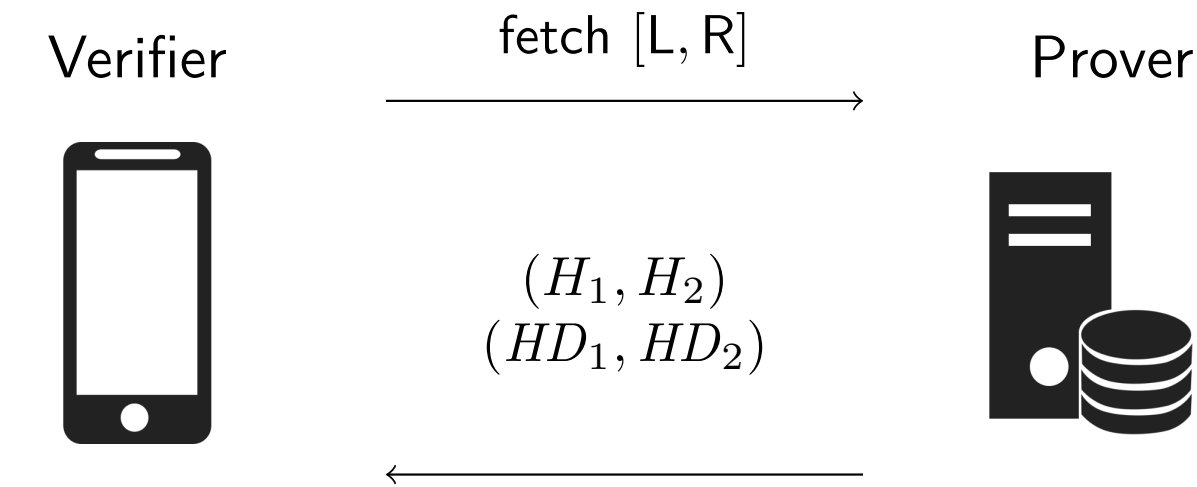
Merkle tree evaluation



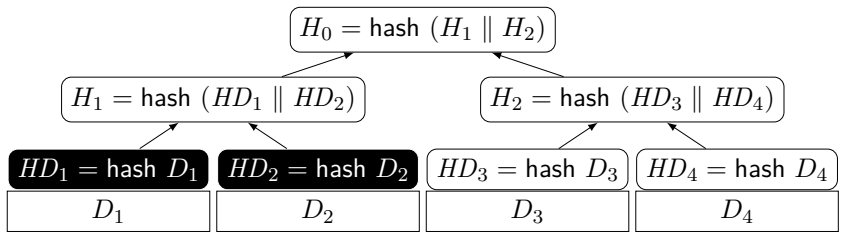
$H_0 = \text{hash } (H_1 \parallel H_2)$



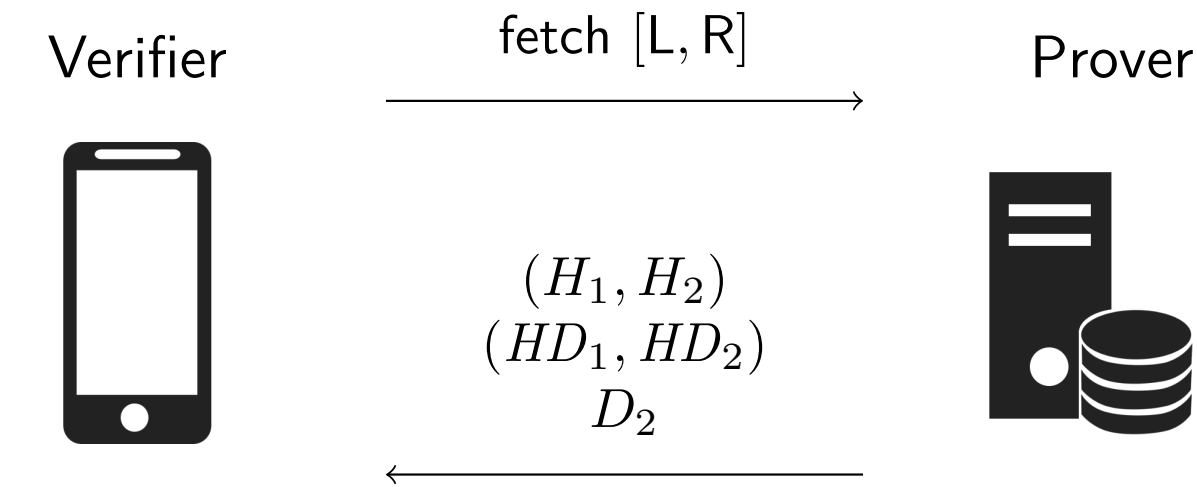
Merkle tree evaluation



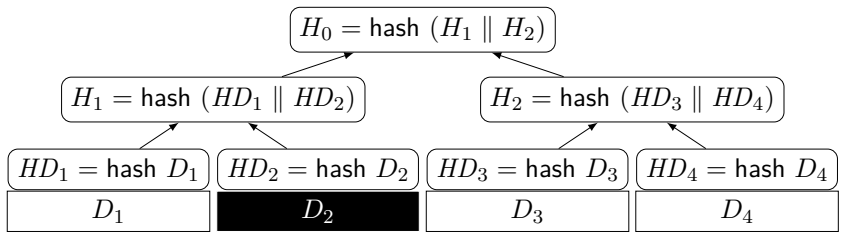
$H_0 = \mathsf{hash}\ (H_1 \parallel H_2)$



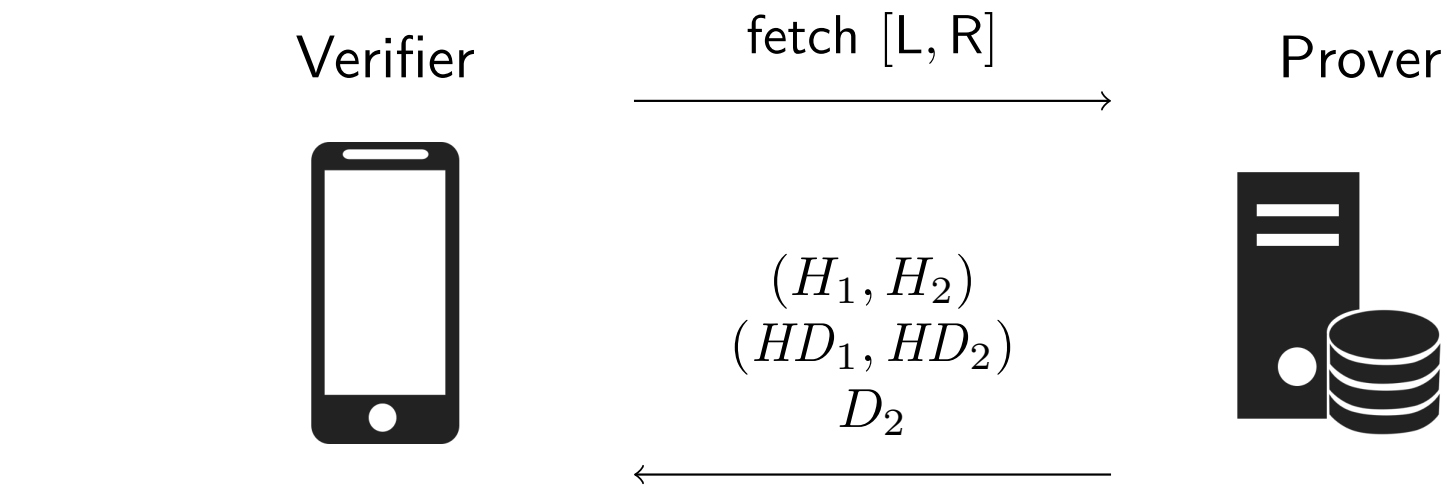
Merkle tree evaluation

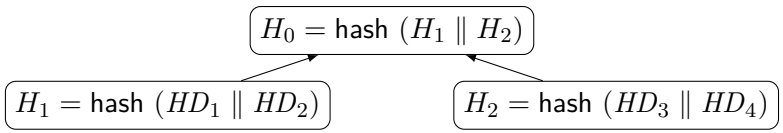


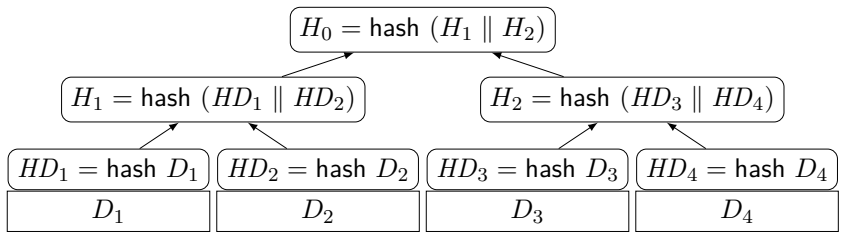
$H_0 = \mathsf{hash}\ (H_1 \parallel H_2)$



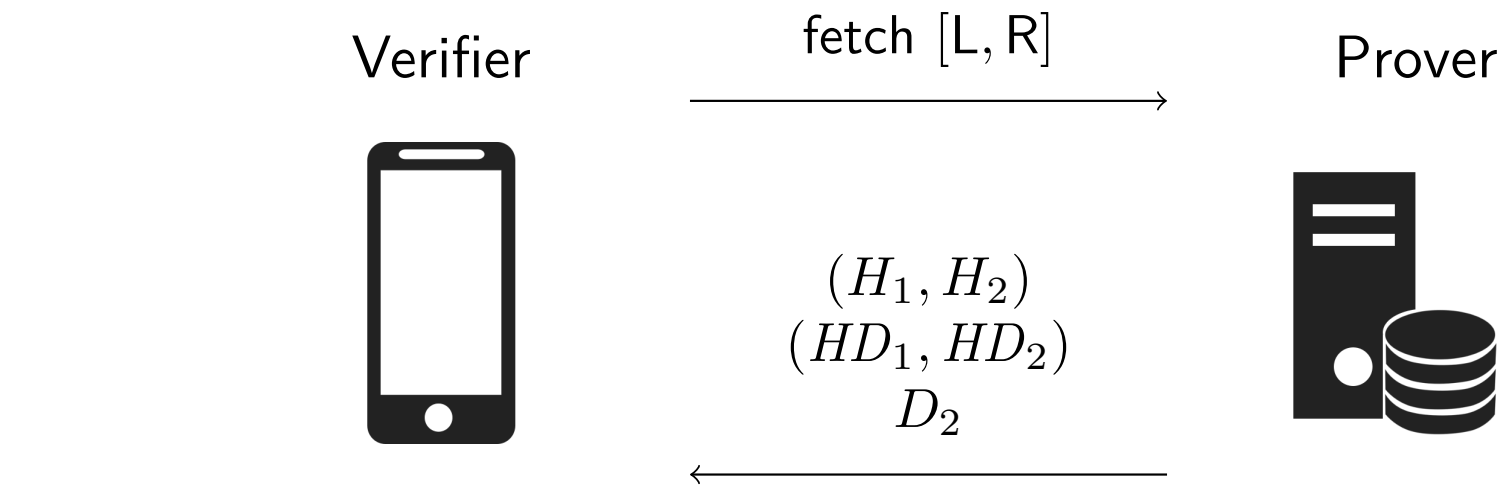
Merkle tree evaluation

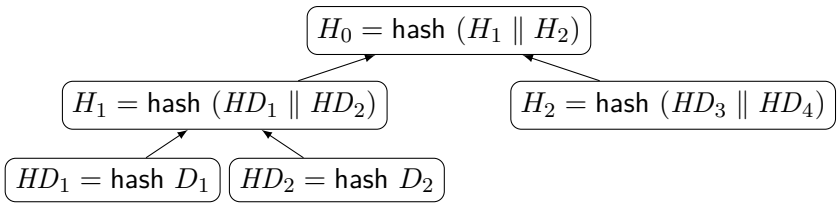


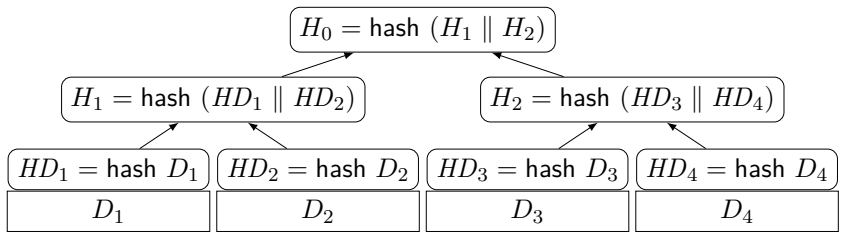




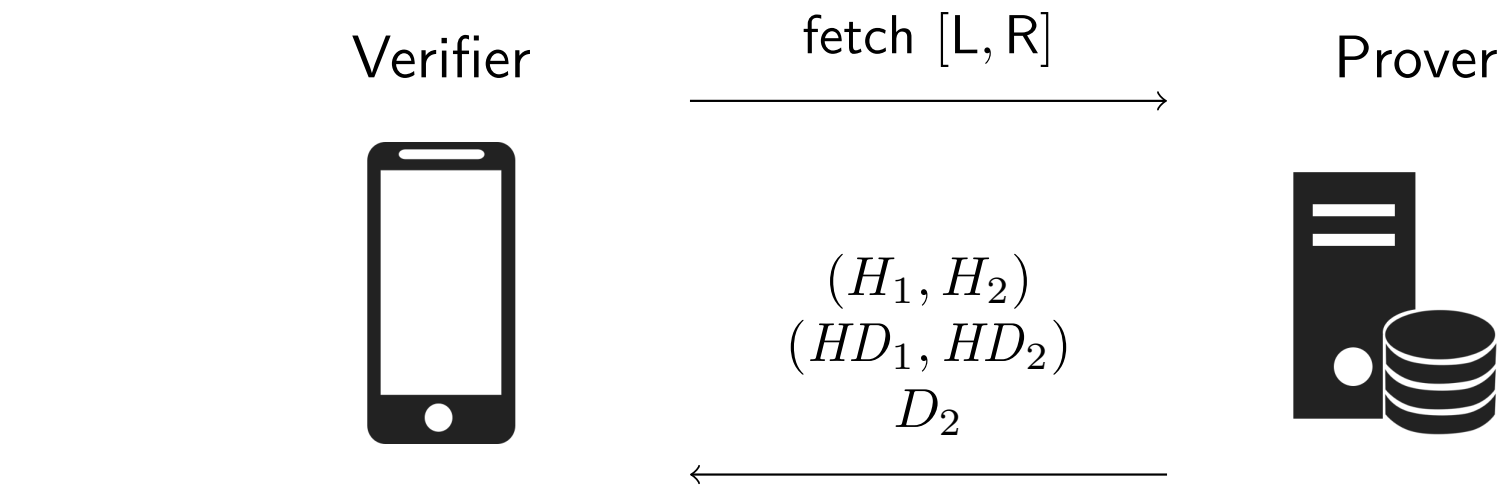
Merkle tree evaluation

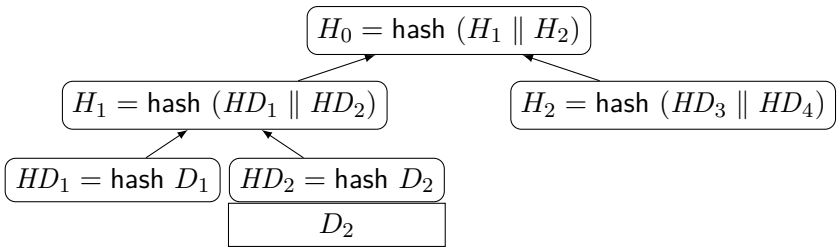


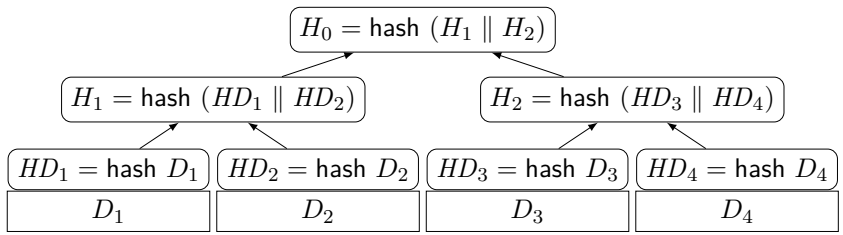




Merkle tree evaluation







Outline

Object of Study

Nominal Isabelle

Formalization

Conclusion

$\lambda\bullet$'s syntax

$$
\begin{array}{rcl}
\text{Types } \tau & ::= & 1 \mid \tau_1 \to \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2 \mid \mu\alpha.\tau \mid \alpha \mid \bullet\tau \\
\text{Values } v & ::= & () \mid x \mid \lambda x.e \mid \mathbf{rec}\ x.\lambda y.e \\
& \mid & \mathbf{inj}_1\ v \mid \mathbf{inj}_2\ v \mid (v_1, v_2) \mid \mathbf{roll}\ v \\
\text{Exprs } e & ::= & v \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \mid v_1\ v_2 \mid \mathbf{case}\ v\ v_0\ v_1 \\
& \mid & \mathbf{prj}_1\ v \mid \mathbf{prj}_2\ v \mid \mathbf{unroll}\ v \mid \mathit{auth}\ v \mid \mathit{unauth}\ v
\end{array}
$$

Figure 5. Syntax for types and terms

$\lambda\bullet$'s syntax

$$
\begin{array}{rcl}
\text{Types } \tau & ::= & 1 \mid \tau_1 \to \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2 \mid \mu\alpha.\tau \mid \alpha \mid \bullet\tau \\
\text{Values } v & ::= & () \mid x \mid \lambda x.e \mid \mathbf{rec}\ x.\lambda y.e \\
& \mid & \mathbf{inj}_1\ v \mid \mathbf{inj}_2\ v \mid (v_1, v_2) \mid \mathbf{roll}\ v \\
\text{Exprs } e & ::= & v \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \mid v_1\ v_2 \mid \mathbf{case}\ v\ v_0\ v_1 \\
& \mid & \mathbf{prj}_1\ v \mid \mathbf{prj}_2\ v \mid \mathbf{unroll}\ v \mid \mathit{auth}\ v \mid \mathit{unauth}\ v
\end{array}
$$

Figure 5. Syntax for types and terms

Bound variables?

Bound variables?

$$\lambda x.x \stackrel{?}{=} \lambda y.y$$

Nominal Isabelle

Nominal Isabelle

nominal_datatype $term$ $=$
  Var var
| Lam $(x :: var)$ $(t :: term)$ **binds** x **in** t
| App $term$ $term$

Nominal Isabelle

nominal_datatype $term$ $=$
  Var var
| Lam $(x :: var)$ $(t :: term)$ **binds** x **in** t
| App $term$ $term$

inductive $value$ $::$ $term \Rightarrow bool$ **where**
  value (Var x)
| value (Lam x e)

Nominal Isabelle - Freshness

Nominal Isabelle - Freshness

$$x \sharp e$$

Nominal Isabelle - Freshness

$$x \sharp e \quad \approx \quad x \notin FV(e)$$

Nominal Isabelle - Freshness

$$x \sharp e \quad \approx \quad x \notin FV(e)$$

$$\neg x \sharp \mathsf{Var}\ x$$

Nominal Isabelle - Freshness

$$x \sharp e \quad \approx \quad x \notin FV(e)$$

$$\neg x \sharp \mathsf{Var}\ x \qquad x \sharp \mathsf{Lam}\ x\ (\mathsf{Var}\ x)$$

Capture-Avoiding Substitution

Capture-Avoiding Substitution

$$\textbf{nominal\_function}\ \mathit{subst} :: [\ldots]\ (\_[\_/\_])\ \textbf{where}$$

$$(\mathsf{Var}\ y)[t'/x] = (\mathsf{if}\ x = y\ \mathsf{then}\ t'\ \mathsf{else}\ \mathsf{Var}\ y)$$

$$\mid\ y \sharp (x, t') \longrightarrow (\mathsf{Lam}\ y\ t)[t'/x] = \mathsf{Lam}\ y\ (t[t'/x])$$

$$\mid\ (\mathsf{App}\ t_1\ t_2)[t'/x] = \mathsf{App}\ (t_1[t'/x])\ (t_2[t'/x])$$

Capture-Avoiding Substitution

$$\textbf{nominal\_function}\ \mathit{subst} :: [\ldots]\ (\_[\_/\_])\ \textbf{where}$$

$$(\mathsf{Var}\ y)[t'/x] = (\mathsf{if}\ x = y\ \mathsf{then}\ t'\ \mathsf{else}\ \mathsf{Var}\ y)$$

$$\mid\ y \sharp (x, t') \longrightarrow (\mathsf{Lam}\ y\ t)[t'/x] = \mathsf{Lam}\ y\ (t[t'/x])$$

$$\mid\ (\mathsf{App}\ t_1\ t_2)[t'/x] = \mathsf{App}\ (t_1[t'/x])\ (t_2[t'/x])$$

Nominal Induction

Nominal Induction

lemma subst_idle : "$y \sharp t \Longrightarrow t[s/y] = t$"
proof (nominal_induct t **avoiding** : s y
rule : term.strong_induct)
...

Nominal Induction

lemma subst_idle : "$y \sharp t \Longrightarrow t[s/y] = t$"
proof (nominal_induct t **avoiding** : s y
rule : term.strong_induct)
...

Nominal Induction

lemma subst_idle : $"y \sharp t \Longrightarrow t[s/y] = t"$
proof (nominal_induct t **avoiding** : s y
rule : term.strong_induct)
...

Additional assumptions in the case for Lam x e:

$$x \sharp s$$
$$x \sharp y$$

Full Syntax

Full Syntax

nominal_datatype *term* =
  Unit
| Var *var*
| Lam (*x* :: *var*) (*t* :: *term*) **binds** *x* **in** *t*
| Rec (*x* :: *var*) (*t* :: *term*) **binds** *x* **in** *t*
| Inj1 *term*
| Inj2 *term*
| Pair *term term*
| Let *term* (*x* :: *var*) (*t* :: *term*) **binds** *x* **in** *t*
| App *term term*
| Case *term term term*
| Prj1 *term*
| Prj2 *term*
| Roll *term*
| Unroll *term*
| Auth *term*
| Unauth *term*
| Hash *hash*
| Hashed *hash term*

nominal_datatype *ty* =
  One
| Fun *ty ty*
| Sum *ty ty*
| Prod *ty ty*
| Mu (*α* :: *tvar*) (*τ* :: *ty*) **binds** *α* **in** *τ*
| Alpha *tvar*
| AuthT *ty*

inductive *value* :: *term* ⇒ *bool* **where**
  value Unit
| value (Var *x*)
| value (Lam *x e*)
| value (Rec *x e*)
| value *v* ⟶ value (Inj1 *v*)
| value *v* ⟶ value (Inj2 *v*)
| value v_1 ∧ value v_2 ⟶ value (Pair v_1 v_2)
| value *v* ⟶ value (Roll *v*)
| value (Hash *h*)
| value (Hashed *h e*)

Full Syntax

nominal_datatype *term* =
  Unit
| Var *var*
| Lam (*x* :: *var*) (*t* :: *term*) **binds** *x* **in** *t*
| Rec (*x* :: *var*) (*t* :: *term*) **binds** *x* **in** *t*
| Inj1 *term*
| Inj2 *term*
| Pair *term term*
| Let *term* (*x* :: *var*) (*t* :: *term*) **binds** *x* **in** *t*
| App *term term*
| Case *term term term*
| Prj1 *term*
| Prj2 *term*
| Roll *term*
| Unroll *term*
| Auth *term*
| Unauth *term*
| Hash *hash*
| Hashed *hash term*

nominal_datatype *ty* =
  One
| Fun *ty ty*
| Sum *ty ty*
| Prod *ty ty*
| Mu (*α* :: *tvar*) (*τ* :: *ty*) **binds** *α* **in** *τ*
| Alpha *tvar*
| AuthT *ty*

inductive *value* :: *term* ⇒ *bool* **where**
  value Unit
| value (Var *x*)
| value (Lam *x e*)
| value (Rec *x e*)
| value *v* ⟶ value (Inj1 *v*)
| value *v* ⟶ value (Inj2 *v*)
| value v_1 ∧ value v_2 ⟶ value (Pair v_1 v_2)
| value *v* ⟶ value (Roll *v*)
| value (Hash *h*)
| value (Hashed *h e*)

Merkle trees in $\lambda\bullet$

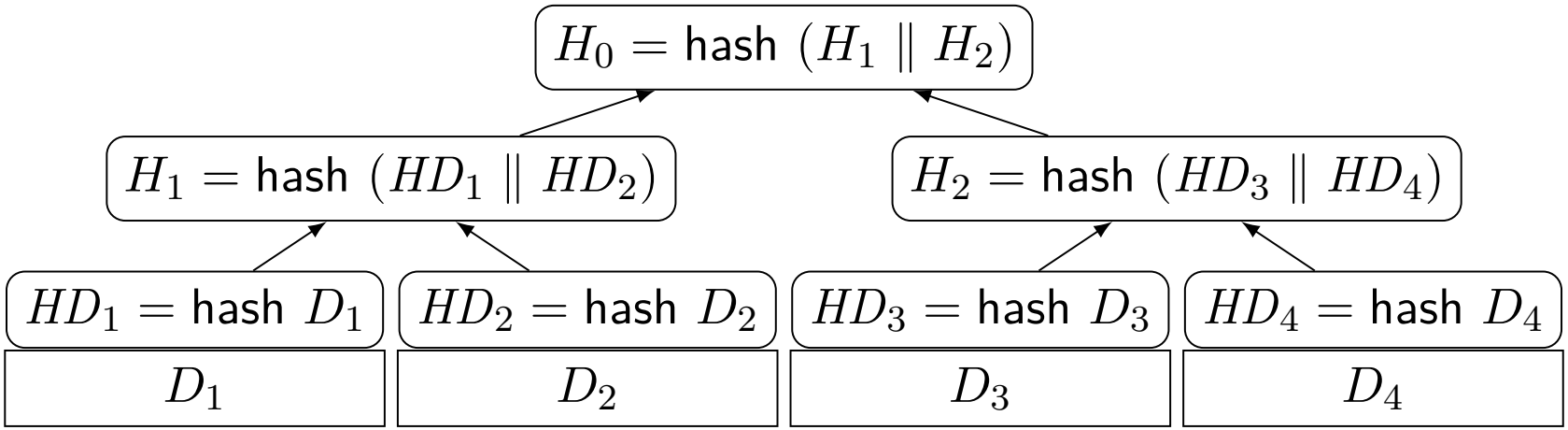
$$\mathbf{type}\ \mathsf{tree}\ =\ \mathsf{Leaf}\ \mathsf{of}\ \mathsf{string} \mid \mathsf{Bin}\ \mathsf{of}\ \ \bullet\,\mathsf{tree} \times \bullet\mathsf{tree}$$

Merkle trees in $\lambda\bullet$

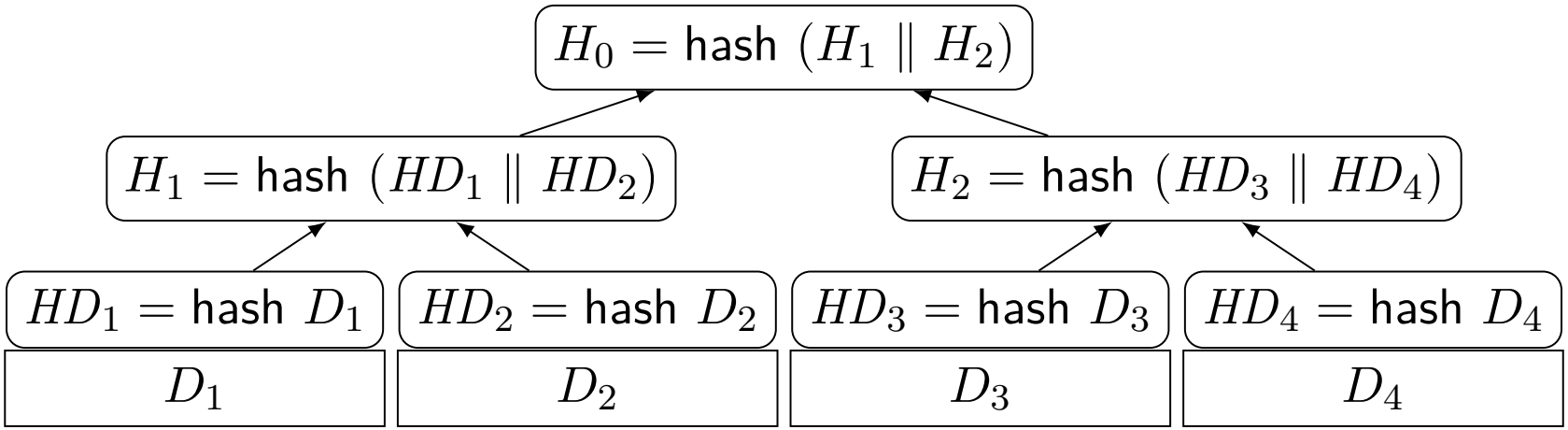
$$\mathbf{type}\ \mathsf{tree}\ =\ \mathsf{Leaf\ of\ string}\ |\ \mathsf{Bin\ of}\ \ \bullet\,\mathsf{tree} \times \bullet\mathsf{tree}$$

$$\mu T.\ \mathsf{string} + (\mathsf{AuthT}\ T \times \mathsf{AuthT}\ T)$$

Merkle trees in $\lambda\bullet$



Merkle trees in $\lambda\bullet$



Prover:

⟨H0, Bin
  ⟨H1, Bin
    ⟨HD1, Leaf D1⟩
    ⟨HD2, Leaf D2⟩⟩
  ⟨H2, Bin
    ⟨HD3, Leaf D3⟩
    ⟨HD4, Leaf D4⟩⟩⟩

Verifier:

H0

Typing Judgment (excerpt)

Typing Judgment (excerpt)

$$\frac{\Gamma[x] = \mathsf{Some}\ \tau}{\Gamma \vdash \mathsf{Var}\ x : \tau} \qquad \frac{x \notin \Gamma \quad \Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \mathsf{Lam}\ x\ e : \mathsf{Fun}\ \tau_1\ \tau_2}$$

$$\frac{\Gamma \vdash e : \mathsf{Fun}\ \tau_1\ \tau_2 \quad \Gamma \vdash e' : \tau_1}{\Gamma \vdash \mathsf{App}\ e\ e' : \tau_2}$$

Typing Judgment (excerpt)

$$\frac{\Gamma[x] = \mathsf{Some}\ \tau}{\Gamma \vdash \mathsf{Var}\ x : \tau} \qquad \frac{x \notin \Gamma \quad \Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \mathsf{Lam}\ x\ e : \mathsf{Fun}\ \tau_1\ \tau_2}$$

$$\frac{\Gamma \vdash e : \mathsf{Fun}\ \tau_1\ \tau_2 \quad \Gamma \vdash e' : \tau_1}{\Gamma \vdash \mathsf{App}\ e\ e' : \tau_2}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{Auth}\ e : \mathsf{AuthT}\ \tau} \qquad \frac{\Gamma \vdash e : \mathsf{AuthT}\ \tau}{\Gamma \vdash \mathsf{Unauth}\ e : \tau}$$

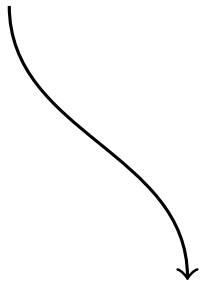
Small-step Semantics

Small-step Semantics

$$\langle \pi_1,\, e_1 \rangle \; m \rightarrow_i \langle \pi_2,\, e_2 \rangle$$

Small-step Semantics

Expression e_1 with proof stream π_1



$$\langle \pi_1,\ e_1 \rangle\ m \rightarrow_i \langle \pi_2,\ e_2 \rangle$$

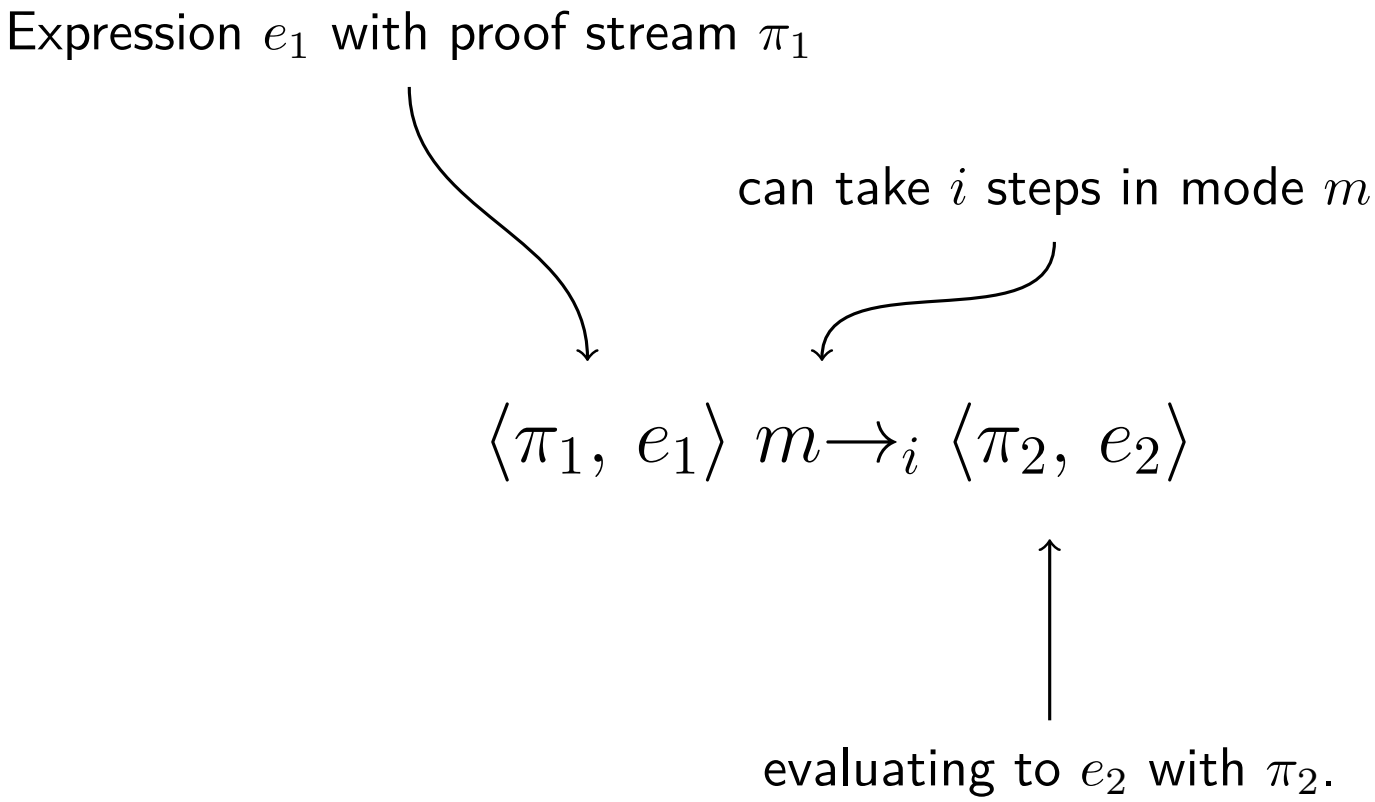
Small-step Semantics

Expression e_1 with proof stream π_1

can take i steps in mode m

$$\langle \pi_1,\ e_1 \rangle\ m{\longrightarrow_i}\ \langle \pi_2,\ e_2 \rangle$$

Small-step Semantics



Proof stream? Mode?

Proof stream? Mode?

type_synonym $proofstream = term\ list$

Proof stream? Mode?

type_synonym $proofstream = term\ list$

datatype $mode =$ I | P | V

I — Idealized computation

P — Prover computation

V — Verifier computation

Small-step Semantics (excerpt)

Small-step Semantics (excerpt)

$$\frac{\langle \pi,\, e_1 \rangle\ m{\to}\ \langle \pi',\, e_1' \rangle}{\langle \pi,\, \mathsf{App}\ e_1\ e_2 \rangle\ m{\to}\ \langle \pi',\, \mathsf{App}\ e_1'\ e_2 \rangle}$$

$$\frac{\mathsf{value}\ v_1 \qquad \langle \pi,\, e_2 \rangle\ m{\to}\ \langle \pi',\, e_2' \rangle}{\langle \pi,\, \mathsf{App}\ v_1\ e_2 \rangle\ m{\to}\ \langle \pi',\, \mathsf{App}\ v_1\ e_2' \rangle}$$

$$\frac{\mathsf{value}\ v \qquad x \sharp (v, \pi)}{\langle \pi,\, \mathsf{App}\ (\mathsf{Lam}\ x\ e)\ v \rangle\ m{\to}\ \langle \pi,\, e[v/x] \rangle}$$

Small-step Semantics (excerpt)

$$\frac{\langle \pi,\, e_1 \rangle\ m{\to}\ \langle \pi',\, e_1' \rangle}{\langle \pi,\, \mathsf{App}\ e_1\ e_2 \rangle\ m{\to}\ \langle \pi',\, \mathsf{App}\ e_1'\ e_2 \rangle} \qquad \frac{\mathsf{value}\ v_1 \qquad \langle \pi,\, e_2 \rangle\ m{\to}\ \langle \pi',\, e_2' \rangle}{\langle \pi,\, \mathsf{App}\ v_1\ e_2 \rangle\ m{\to}\ \langle \pi',\, \mathsf{App}\ v_1\ e_2' \rangle}$$

$$\frac{\mathsf{value}\ v \qquad x \sharp (v, \pi)}{\langle \pi,\, \mathsf{App}\ (\mathsf{Lam}\ x\ e)\ v \rangle\ m{\to}\ \langle \pi,\, e[v/x] \rangle}$$

$$\frac{}{\langle \pi,\, e \rangle\ m{\to_0}\ \langle \pi,\, e \rangle}$$

$$\frac{\langle \pi_1,\, e_1 \rangle\ m{\to_i}\ \langle \pi_2,\, e_2 \rangle \qquad \langle \pi_2,\, e_2 \rangle\ m{\to}\ \langle \pi_3,\, e_3 \rangle}{\langle \pi_1,\, e_1 \rangle\ m{\to_{i+1}}\ \langle \pi_3,\, e_3 \rangle}$$

Small-step Semantics (excerpt, continued)

Small-step Semantics (excerpt, continued)

$$\frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Auth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Auth}\; e'\rangle} \qquad \frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Unauth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Unauth}\; e'\rangle}$$

Small-step Semantics (excerpt, continued)

$$\frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Auth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Auth}\; e'\rangle} \qquad \frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Unauth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Unauth}\; e'\rangle}$$

$$\frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{I}{\rightarrow}\; \langle \pi,\, v\rangle} \qquad \frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Unauth}\; v\rangle \; \mathsf{I}{\rightarrow}\; \langle \pi,\, v\rangle}$$

Small-step Semantics (excerpt, continued)

$$\frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Auth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Auth}\; e'\rangle} \qquad \frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Unauth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Unauth}\; e'\rangle}$$

$$\frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{I}{\rightarrow}\; \langle \pi,\, v\rangle} \qquad \frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Unauth}\; v\rangle \; \mathsf{I}{\rightarrow}\; \langle \pi,\, v\rangle}$$

$$\frac{\mathsf{closed}\; (\!|v|\!) \qquad \mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{P}{\rightarrow}\; \langle \pi,\, \mathsf{Hashed}\; (\mathsf{hash}\; (\!|v|\!))\; v\rangle}$$

Small-step Semantics (excerpt, continued)

$$\frac{\langle \pi,\, e\rangle \; m{\to}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Auth}\; e\rangle \; m{\to}\; \langle \pi',\, \mathsf{Auth}\; e'\rangle} \qquad \frac{\langle \pi,\, e\rangle \; m{\to}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Unauth}\; e\rangle \; m{\to}\; \langle \pi',\, \mathsf{Unauth}\; e'\rangle}$$

$$\frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{I}{\to}\; \langle \pi,\, v\rangle} \qquad \frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Unauth}\; v\rangle \; \mathsf{I}{\to}\; \langle \pi,\, v\rangle}$$

$$\frac{\mathsf{closed}\; (\!|v|\!) \qquad \mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{P}{\to}\; \langle \pi,\, \mathsf{Hashed}\; (\mathsf{hash}\; (\!|v|\!))\; v\rangle} \qquad \frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Unauth}\; (\mathsf{Hashed}\; h\; v)\rangle \; \mathsf{P}{\to}\; \langle \pi \;@\; [(\!|v|\!)],\, v\rangle}$$

Small-step Semantics (excerpt, continued)

$$\frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Auth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Auth}\; e'\rangle} \qquad \frac{\langle \pi,\, e\rangle \; m{\rightarrow}\; \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Unauth}\; e\rangle \; m{\rightarrow}\; \langle \pi',\, \mathsf{Unauth}\; e'\rangle}$$

$$\frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{I}{\rightarrow}\; \langle \pi,\, v\rangle} \qquad \frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Unauth}\; v\rangle \; \mathsf{I}{\rightarrow}\; \langle \pi,\, v\rangle}$$

$$\frac{\mathsf{closed}\; (\!|v|\!) \qquad \mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{P}{\rightarrow}\; \langle \pi,\, \mathsf{Hashed}\; (\mathsf{hash}\; (\!|v|\!))\; v\rangle} \qquad \frac{\mathsf{value}\; v}{\langle \pi,\, \mathsf{Unauth}\; (\mathsf{Hashed}\; h\; v)\rangle \; \mathsf{P}{\rightarrow}\; \langle \pi\; @\; [(\!|v|\!)],\, v\rangle}$$

$$\frac{\mathsf{closed}\; v \qquad \mathsf{value}\; v}{\langle \pi,\, \mathsf{Auth}\; v\rangle \; \mathsf{V}{\rightarrow}\; \langle \pi,\, \mathsf{Hash}\; (\mathsf{hash}\; v)\rangle}$$

Small-step Semantics (excerpt, continued)

$$\frac{\langle \pi,\, e\rangle\ m{\rightarrow}\ \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Auth}\ e\rangle\ m{\rightarrow}\ \langle \pi',\, \mathsf{Auth}\ e'\rangle} \qquad \frac{\langle \pi,\, e\rangle\ m{\rightarrow}\ \langle \pi',\, e'\rangle}{\langle \pi,\, \mathsf{Unauth}\ e\rangle\ m{\rightarrow}\ \langle \pi',\, \mathsf{Unauth}\ e'\rangle}$$

$$\frac{\mathsf{value}\ v}{\langle \pi,\, \mathsf{Auth}\ v\rangle\ \mathsf{I}{\rightarrow}\ \langle \pi,\, v\rangle} \qquad \frac{\mathsf{value}\ v}{\langle \pi,\, \mathsf{Unauth}\ v\rangle\ \mathsf{I}{\rightarrow}\ \langle \pi,\, v\rangle}$$

$$\frac{\mathsf{closed}\ (\!|v|\!) \qquad \mathsf{value}\ v}{\langle \pi,\, \mathsf{Auth}\ v\rangle\ \mathsf{P}{\rightarrow}\ \langle \pi,\, \mathsf{Hashed}\ (\mathsf{hash}\ (\!|v|\!))\ v\rangle} \qquad \frac{\mathsf{value}\ v}{\langle \pi,\, \mathsf{Unauth}\ (\mathsf{Hashed}\ h\ v)\rangle\ \mathsf{P}{\rightarrow}\ \langle \pi\ @\ [(\!|v|\!)],\, v\rangle}$$

$$\frac{\mathsf{closed}\ v \qquad \mathsf{value}\ v}{\langle \pi,\, \mathsf{Auth}\ v\rangle\ \mathsf{V}{\rightarrow}\ \langle \pi,\, \mathsf{Hash}\ (\mathsf{hash}\ v)\rangle} \qquad \frac{\mathsf{closed}\ s_0 \qquad \mathsf{hash}\ s_0 = h}{\langle s_0 \# \pi,\, \mathsf{Unauth}\ (\mathsf{Hash}\ h)\rangle\ \mathsf{V}{\rightarrow}\ \langle \pi,\, s_0\rangle}$$

Merkle trees in $\lambda\bullet$

$$\mathbf{type}\ \text{tree}\ =\ \text{Leaf of string} \mid \text{Bin of}\ \ \bullet\,\text{tree} \times \bullet\text{tree}$$

Example adapted from Miller et al.

Using some syntactic sugar

Merkle trees in $\lambda\bullet$

```
type tree = Leaf of string | Bin of • tree × •tree
type bit = L | R

let rec fetch (idx : bit list) (t : •tree) : string =
  case (idx, unauth t) of
    | ([], Leaf a) → a
    | (L # idx, Bin(l, _)) → fetch idx l
    | (R # idx, Bin(_, r)) → fetch idx r
```

Example adapted from Miller et al.

Using some syntactic sugar

Hash Function with Nominal Isabelle

Hash Function with Nominal Isabelle

$$\forall p.\ p \bullet \mathsf{hash}\ e = \mathsf{hash}\ (p \bullet e)$$

Hash Function with Nominal Isabelle

$$\forall p. \qquad \mathsf{hash}\ e = \mathsf{hash}\ (p \bullet e)$$

Hash Function with Nominal Isabelle

$$\forall p. \qquad \mathsf{hash}\ e = \mathsf{hash}\ (p \bullet e)$$

$$\mathsf{hash}\ (\mathsf{Var}\ x) = \mathsf{hash}\ (\mathsf{Var}\ y)$$

Hash Function with Nominal Isabelle

$$\forall p. \qquad \mathsf{hash}\ e = \mathsf{hash}\ (p \bullet e)$$

$$\mathsf{hash}\ (\mathsf{Var}\ x) = \mathsf{hash}\ (\mathsf{Var}\ y)$$

Okay, as long as we use hash on closed expressions.

Lemma 1: Type Soundness

Lemma 1: Type Soundness

If $\varnothing \vdash e : \tau$

Lemma 1: Type Soundness

If $\varnothing \vdash e : \tau$

then value e

Lemma 1: Type Soundness

$$
\begin{array}{ll}
\text{If} & \varnothing \vdash e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle \mathsf{I}\!\!\rightarrow \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau
\end{array}
$$

Lemma 1: Type Soundness

$$\begin{array}{ll} \text{If} & \varnothing \vdash e : \tau \\ \text{then} & \mathsf{value}\ e \\ \text{or} & \exists e'.\ \langle [], e \rangle\ \mathsf{I}{\longrightarrow}\ \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau \end{array}$$

$$\varnothing \vdash \mathsf{Unit} : \mathsf{One}$$

Lemma 1: Type Soundness

$$
\begin{array}{ll}
\text{If} & \varnothing \vdash e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle\ \mathsf{I}{\longrightarrow}\ \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau
\end{array}
$$

$$\varnothing \vdash \mathsf{Auth\ Unit} : \mathsf{AuthT\ One}$$

Lemma 1: Type Soundness

$$
\begin{array}{ll}
\text{If} & \varnothing \vdash e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle \mathsf{I}\!\!\rightarrow \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau
\end{array}
$$

$\varnothing \vdash$ Auth Unit : AuthT One

¬value (Auth Unit)

Lemma 1: Type Soundness

$$
\begin{array}{ll}
\text{If} & \varnothing \vdash e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle\ \mathsf{I}{\rightarrow}\ \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau
\end{array}
$$

$\varnothing \vdash$ Auth Unit : AuthT One

$\neg$value (Auth Unit)

$\langle [],$ Auth Unit$\rangle\ \mathsf{I}{\rightarrow}\ \langle [],$ Unit$\rangle$

Lemma 1: Type Soundness

$$
\begin{array}{ll}
\text{If} & \varnothing \vdash e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle\ \mathsf{I}{\rightarrow}\ \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau
\end{array}
$$

$\varnothing \vdash \mathsf{Auth\ Unit} : \mathsf{AuthT\ One}$

$\neg\mathsf{value}\ (\mathsf{Auth\ Unit})$

$\langle [], \mathsf{Auth\ Unit} \rangle\ \mathsf{I}{\rightarrow}\ \langle [], \mathsf{Unit} \rangle$

$\varnothing \vdash \mathsf{Unit} : \mathsf{One}$

Lemma 1: Type Soundness

If $\varnothing \vdash e : \tau$
then value e
or $\exists e'.\ \langle [], e \rangle \mathsf{I}\!\rightarrow \langle [], e' \rangle \wedge \varnothing \vdash e' : \tau$

$\varnothing \vdash$ Auth Unit : AuthT One

¬value (Auth Unit)

$\langle [],$ Auth Unit$\rangle \mathsf{I}\!\rightarrow \langle [],$ Unit$\rangle$

$\varnothing \vdash$ Unit : One

Type Soundness?

Miller et al.:
"for mode I, authenticated values of type $\bullet\tau$ [i.e., AuthT τ]
are merely values of type τ"

Type Soundness?

Miller et al.:
"for mode I, authenticated values of type $\bullet\tau$ [i.e., AuthT τ]
are merely values of type τ"

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{Auth}\ e : \mathsf{AuthT}\ \tau} \qquad \frac{\Gamma \vdash e : \mathsf{AuthT}\ \tau}{\Gamma \vdash \mathsf{Unauth}\ e : \tau}$$

Type Soundness?

Miller et al.:
"for mode I, authenticated values of type $\bullet\tau$ [i.e., AuthT τ]
are merely values of type τ"

$$\frac{\Gamma \vdash_W e : \tau}{\Gamma \vdash_W \mathsf{Auth}\ e : \tau} \qquad \frac{\Gamma \vdash_W e : \tau}{\Gamma \vdash_W \mathsf{Unauth}\ e : \tau}$$

Type Soundness?

Miller et al.:
"for mode I, authenticated values of type $\bullet\tau$ [i.e., AuthT τ] are merely values of type τ"

$$\frac{\Gamma \vdash_W e : \tau}{\Gamma \vdash_W \mathsf{Auth}\ e : \tau} \qquad \frac{\Gamma \vdash_W e : \tau}{\Gamma \vdash_W \mathsf{Unauth}\ e : \tau}$$

However: $\varnothing \vdash_W \mathsf{Unauth}\ \mathsf{Unit} : \mathsf{One}$

Type Soundness

$$
\begin{array}{ll}
\text{If} & \varnothing \vdash_W e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle \ \mathsf{I}\!\!\rightarrow \langle [], e' \rangle \wedge \varnothing \vdash_W e' : \tau
\end{array}
$$

Type Soundness

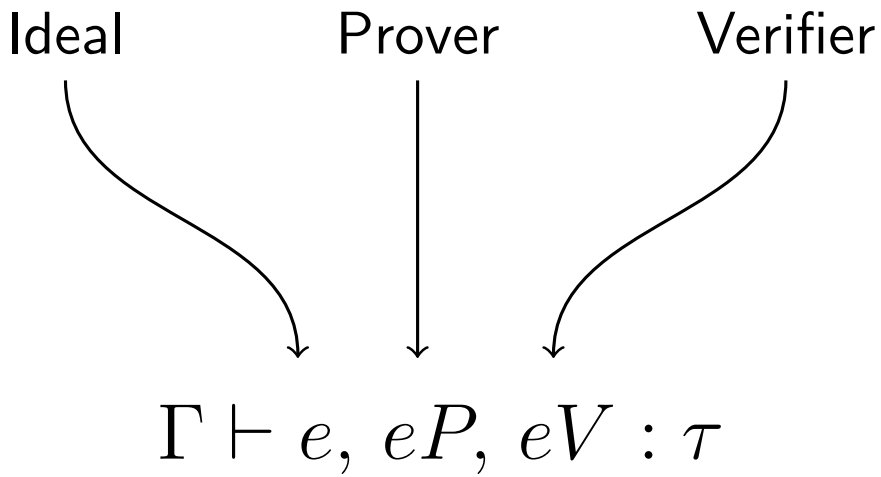
$$
\begin{array}{ll}
\text{If} & \varnothing \vdash_W e : \tau \\
\text{then} & \mathsf{value}\ e \\
\text{or} & \exists e'.\ \langle [], e \rangle\ \mathsf{I}\!\!\rightarrow \langle [], e' \rangle \wedge \varnothing \vdash_W e' : \tau
\end{array}
$$

$$\Gamma \vdash e : \tau \longrightarrow \mathsf{fmmap\ erase}\ \Gamma \vdash_W e : \mathsf{erase}\ \tau$$

Agreement

$$\Gamma \vdash e,\ eP,\ eV : \tau$$

Agreement



Agreement

Agreement

$$\frac{\Gamma[x] = \mathsf{Some}\ \tau}{\Gamma \vdash \mathsf{Var}\ x,\ \mathsf{Var}\ x,\ \mathsf{Var}\ x : \tau}$$

$$\frac{x \sharp \Gamma \qquad \Gamma[x \mapsto \tau_1] \vdash e,\ eP,\ eV : \tau_2}{\Gamma \vdash \mathsf{Lam}\ x\ e,\ \mathsf{Lam}\ x\ eP,\ \mathsf{Lam}\ x\ eV : \mathsf{Fun}\ \tau_1\ \tau_2}$$

$$\frac{\Gamma \vdash e_1,\ eP_1,\ eV_1 : \mathsf{Fun}\ \tau_1\ \tau_2 \qquad \Gamma \vdash e_2,\ eP_2,\ eV_2 : \tau_1}{\Gamma \vdash \mathsf{App}\ e_1\ e_2,\ \mathsf{App}\ eP_1\ eP_2,\ \mathsf{App}\ eV_1\ eV_2 : \tau_2}$$

$$\frac{\Gamma \vdash e,\ eP,\ eV : \tau}{\Gamma \vdash \mathsf{Auth}\ e,\ \mathsf{Auth}\ eP,\ \mathsf{Auth}\ eV : \mathsf{AuthT}\ \tau} \qquad \frac{\Gamma \vdash e,\ eP,\ eV : \mathsf{AuthT}\ \tau}{\Gamma \vdash \mathsf{Unauth}\ e,\ \mathsf{Unauth}\ eP,\ \mathsf{Unauth}\ eV : \tau}$$

Agreement (continued)

Agreement (continued)

$$\frac{\mathsf{value}\ v \qquad \mathsf{value}\ vP \qquad \varnothing \vdash v,\ vP,\ (\!|vP|\!) : \tau \qquad \mathsf{hash}\ (\!|vP|\!) = h}{\Gamma \vdash v,\ \mathsf{Hashed}\ h\ vP,\ \mathsf{Hash}\ h : \mathsf{AuthT}\ \tau}$$

Agreement (continued)

$$\frac{\mathsf{value}\ v \qquad \mathsf{value}\ vP \qquad \varnothing \vdash v,\ vP,\ (\!|vP|\!) : \tau \qquad \mathsf{hash}\ (\!|vP|\!) = h}{\Gamma \vdash v,\ \mathsf{Hashed}\ h\ vP,\ \mathsf{Hash}\ h : \mathsf{AuthT}\ \tau}$$

$$\frac{\mathsf{value}\ v}{\langle \pi,\ \mathsf{Auth}\ v \rangle\ \mathsf{I}{\rightarrow}\ \langle \pi,\ v \rangle}$$

$$\frac{\mathsf{closed}\ (\!|v|\!) \qquad \mathsf{value}\ v}{\langle \pi,\ \mathsf{Auth}\ v \rangle\ \mathsf{P}{\rightarrow}\ \langle \pi,\ \mathsf{Hashed}\ (\mathsf{hash}\ (\!|v|\!))\ v \rangle}$$

$$\frac{\mathsf{closed}\ v \qquad \mathsf{value}\ v}{\langle \pi,\ \mathsf{Auth}\ v \rangle\ \mathsf{V}{\rightarrow}\ \langle \pi,\ \mathsf{Hash}\ (\mathsf{hash}\ v) \rangle}$$

Correctness and Security

Given $\varnothing \vdash e$, eP, $eV : \tau$, if

Correctness and Security

Given $\varnothing \vdash e, eP, eV : \tau$, if

1. *Correctness:* e takes a step
2. *Security:* eV takes a step
3. *Remark 1:* eP takes a step

what happens with the other two expressions?

Single-step Security

$$\text{If} \quad \varnothing \vdash e,\ eP,\ eV : \tau$$

$$\text{and} \quad \langle \pi_A,\ eV \rangle \ \mathsf{V}\!\!\rightarrow \langle \pi',\ eV' \rangle$$

Single-step Security

$$\text{If} \quad \varnothing \vdash e,\ eP,\ eV : \tau$$

$$\text{and} \quad \langle \pi_A,\ eV \rangle\ \mathsf{V}{\rightarrow}\ \langle \pi',\ eV' \rangle$$

$$\text{then for some } e', eP', \pi$$

$$\langle [],\ e \rangle\ \mathsf{I}{\rightarrow}\ \langle [],\ e' \rangle$$

$$\wedge\ \forall \pi_P.\ \langle \pi_P,\ eP \rangle\ \mathsf{P}{\rightarrow}\ \langle \pi_P\ @\ \pi,\ eP' \rangle$$

$$\wedge\ \varnothing \vdash e',\ eP',\ eV' : \tau \wedge \pi_A = \pi\ @\ \pi'$$

Single-step Security

$$\text{If} \quad \varnothing \vdash e,\ eP,\ eV : \tau$$

$$\text{and} \quad \langle \pi_A,\ eV \rangle\ \mathsf{V}{\rightarrow}\ \langle \pi',\ eV' \rangle$$

$$\text{then for some } e', eP', \pi$$

$$\langle [],\ e \rangle\ \mathsf{I}{\rightarrow}\ \langle [],\ e' \rangle$$

$$\wedge\ \forall \pi_P.\ \langle \pi_P,\ eP \rangle\ \mathsf{P}{\rightarrow}\ \langle \pi_P\ @\ \pi,\ eP' \rangle$$

$$\wedge\ \varnothing \vdash e',\ eP',\ eV' : \tau \wedge \pi_A = \pi\ @\ \pi'$$

$$\text{or}$$

$$\exists s, s'.\ \pi = [s] \wedge \pi_A = [s']\ @\ \pi' \wedge s \neq s' \wedge$$
$$\mathsf{hash}\ s = \mathsf{hash}\ s' \wedge \mathsf{closed}\ s \wedge \mathsf{closed}\ s'$$

Multi-step Security

If $\varnothing \vdash e,\ eP,\ eV : \tau$

and $\langle \pi_A,\ eV \rangle\ \mathsf{V}{\rightarrow_i}\ \langle \pi',\ eV' \rangle$

Multi-step Security

If $\varnothing \vdash e,\ eP,\ eV : \tau$

and $\langle \pi_A,\ eV \rangle\ \mathsf{V}{\rightarrow_i}\ \langle \pi',\ eV' \rangle$

then for some e', eP', π

$$\langle [],\ e \rangle\ \mathsf{I}{\rightarrow_i}\ \langle [],\ e' \rangle \wedge \langle [],\ eP \rangle\ \mathsf{P}{\rightarrow_i}\ \langle \pi,\ eP' \rangle \wedge$$
$$\pi_A = \pi\ @\ \pi' \wedge \varnothing \vdash e',\ eP',\ eV' : \tau$$

Multi-step Security

If $\varnothing \vdash e,\ eP,\ eV : \tau$

and $\langle \pi_A,\ eV \rangle\ \mathsf{V}{\rightarrow_i}\ \langle \pi',\ eV' \rangle$

then for some e', eP', π

$$\langle [],\ e \rangle\ \mathsf{I}{\rightarrow_i}\ \langle [],\ e' \rangle \wedge \langle [],\ eP \rangle\ \mathsf{P}{\rightarrow_i}\ \langle \pi,\ eP' \rangle \wedge$$
$$\pi_A = \pi\ @\ \pi' \wedge \varnothing \vdash e',\ eP',\ eV' : \tau$$

or for some eP', j, π_0, s, s'

$$j \leq i \wedge \langle [],\ eP \rangle\ \mathsf{P}{\rightarrow_j}\ \langle \pi_0\ @\ [s],\ eP' \rangle \wedge$$
$$\pi_A = \pi_0\ @\ [s']\ @\ \pi' \wedge s \neq s' \wedge$$
$$\mathsf{hash}\ s = \mathsf{hash}\ s' \wedge \mathsf{closed}\ s \wedge \mathsf{closed}\ s'$$

Multi-step Security

If $\varnothing \vdash e,\ eP,\ eV : \tau$

and $\langle \boldsymbol{\pi_A},\ eV\rangle\ \mathsf{V}{\rightarrow_i}\ \langle \boldsymbol{\pi'},\ eV'\rangle$

then for some e', eP', π

$$\langle [],\ e\rangle\ \mathsf{I}{\rightarrow_i}\ \langle [],\ e'\rangle \wedge \langle [],\ eP\rangle\ \mathsf{P}{\rightarrow_i}\ \langle \pi,\ eP'\rangle \wedge$$
$$\pi_A = \pi\ @\ \pi' \wedge \varnothing \vdash e',\ eP',\ eV' : \tau$$

or for some eP', j, π_0, s, s'

$$j \leq i \wedge \langle [],\ eP\rangle\ \mathsf{P}{\rightarrow_j}\ \langle \pi_0\ @\ [s],\ eP'\rangle \wedge$$
$$\boldsymbol{\pi_A = \pi_0\ @\ [s']\ @\ \pi'} \wedge s \neq s' \wedge$$
$$\mathsf{hash}\ s = \mathsf{hash}\ s' \wedge \mathsf{closed}\ s \wedge \mathsf{closed}\ s'$$

Multi-step Security

If $\varnothing \vdash e,\ eP,\ eV : \tau$

and $\langle \boldsymbol{\pi_A},\ eV\rangle\ \mathsf{V}{\rightarrow_i}\ \langle \boldsymbol{\pi'},\ eV'\rangle$

then for some e', eP', π

$$\langle [],\ e\rangle\ \mathsf{I}{\rightarrow_i}\ \langle [],\ e'\rangle \wedge \langle [],\ eP\rangle\ \mathsf{P}{\rightarrow_i}\ \langle \pi,\ eP'\rangle \wedge$$
$$\pi_A = \pi\ @\ \pi' \wedge \varnothing \vdash e',\ eP',\ eV' : \tau$$

or for some eP', j, π_0, s, s'

$$j \leq i \wedge \langle [],\ eP\rangle\ \mathsf{P}{\rightarrow_j}\ \langle \pi_0\ @\ [s],\ eP'\rangle \wedge$$
$$\boldsymbol{\pi_A = \pi_0\ @\ [s']\ @\ \pi'} \wedge s \neq s' \wedge$$
$$\mathsf{hash}\ s = \mathsf{hash}\ s' \wedge \mathsf{closed}\ s \wedge \mathsf{closed}\ s'$$

Does the verifier stop after encountering a hash collision?

Multi-step Security

If $\varnothing \vdash e,\ eP,\ eV : \tau$

and $\langle \boldsymbol{\pi_A},\ eV \rangle\ \mathsf{V}{\rightarrow_i}\ \langle \boldsymbol{\pi'},\ eV' \rangle$

then for some $e',\ eP',\ \pi$

$$\langle [],\ e \rangle\ \mathsf{I}{\rightarrow_i}\ \langle [],\ e' \rangle \wedge \langle [],\ eP \rangle\ \mathsf{P}{\rightarrow_i}\ \langle \pi,\ eP' \rangle \wedge$$
$$\pi_A = \pi\ @\ \pi' \wedge \varnothing \vdash e',\ eP',\ eV' : \tau$$

or for some $eP',\ j,\ \pi_0,\ s,\ s'$

$$j \leq i \wedge \langle [],\ eP \rangle\ \mathsf{P}{\rightarrow_j}\ \langle \pi_0\ @\ [s],\ eP' \rangle \wedge$$
$$\boldsymbol{\pi_A = \pi_0\ @\ [s']\ @\ \pi'_0\ @\ \pi'} \wedge s \neq s' \wedge$$
$$\mathsf{hash}\ s = \mathsf{hash}\ s' \wedge \mathsf{closed}\ s \wedge \mathsf{closed}\ s'$$

Does the verifier stop after encountering a hash collision?

Conclusion

Conclusion

Complete formalization of $\lambda\bullet$ in Isabelle/HOL

Conclusion

Complete formalization of $\lambda\bullet$ in Isabelle/HOL

Corrected mistakes and oversights

Conclusion

Complete formalization of $\lambda\bullet$ in Isabelle/HOL

Corrected mistakes and oversights

Formalization is concise, 3500 lines

Conclusion

Complete formalization of $\lambda\bullet$ in Isabelle/HOL

Corrected mistakes and oversights

Formalization is concise, 3500 lines

Conciseness and natural presentation due to Nominal

Conclusion

Complete formalization of $\lambda\bullet$ in Isabelle/HOL

Corrected mistakes and oversights

Formalization is concise, 3500 lines

Conciseness and natural presentation due to Nominal

Not executable

Generic Authenticated Data Structures, Formally

Matthias Brun Dmitriy Traytel

ETH zürich

2019-09-11, ITP'19