Expressive Authorization Policies using Computation Principals

Anitha Gollamudi

UMASS LOWELL

Stephen Chong

HARVARD
Homer can access nuclear_data
Homer can access nuclear_data

Homer trusts Carl
Can Carl Access Nuclear Data?

Homer can access nuclear_data
Homer trusts Carl
Authorization Logic

Principled reasoning about authorization decisions

Carl speaks for Homer

∀P, P speaks for Homer → P can access nuclear_data

Carl can access nuclear_data
Authorization Logic

Principled reasoning about authorization decisions

Carl *speaks for* Homer

∀P, P *speaks for* Homer $\Rightarrow$ P can access nuclear_data

Homer trusts/delegates to Carl

Carl can access nuclear_data
Principled reasoning about authorization decisions

\[ \forall P, P \text{ speaks for } \text{Homer} \Rightarrow P \text{ can access nuclear data} \]
Authorization Logic

Principals play a central role

Carl *speaks for* Homer

∀P, P *speaks for* Homer ⇒ P can access nuclear_data

Carl can access nuclear_data
Authorization Logic

Principals play a central role

Carl speaks for Homer

∀P, P speaks for Homer ⇒ P can access nuclear_data

Carl can access nuclear_data
What are Principals?
What are Principals?

- Entities that can express statements about access control policies

```bash
$ whoami
root
```
What are Principals?

- Entities that can express statements about access control policies

- Examples
  - Users
  - Public keys
  - OS processes
  - Secure channels

```
$ whoami
root
```
What are Principals?

• Entities that can express statements about access control policies

• Examples
  ‣ Users
  ‣ Public keys
  ‣ OS processes
  ‣ Secure channels

• Atomic Principals
Computation: Missing Piece

- Programs or Computations can also express statements about access control policies
- E.g. Program {P} says “Lenny can access nuclear_data on Tuesday”
Computations: Missing Piece

• Programs or Computations can also express statements about access control policies

• E.g. Program \{P\} says “Lenny can access nuclear_data on Tuesday”
Principals representing computations are Computation Principals
Examples of Computation Principals
Examples of Computation Principals

- Smart Contracts
- Trusted Execution Environments
- Mobile code
- eBPF programs

Diagram:
- Process
- Syscall
- eBPF Verifier
- JIT Compiler
- Linux Kernel
- eBPF programs
- TEE
- Smart Contracts
Existing Authorization Logics

Computations

$ whoami
root

No special treatment
But Computation Principals are Distinct
But Computation Principals are Distinct

Structure

Semantics

Verified

Analyzed

Computations
Coal: Authorization logic that distinguishes computation principals from other principals
Express Trust Directly in a Computation

Homer trusts \{P\}
Express Trust Directly in a Computation

Homer trusts \{P\}
Express Trust Directly in a Computation

Homer trusts \( \{P\} \)
Express Trust Directly in a Computation

Homer trusts \( \{P\} \) if \( \{P\} \) is verified to be secure (e.g., differentially private)
Challenge: How to Represent a Computation Principal

Can I use the **Hash Digest** of the computation?
Why Hash Representation is Not Suitable

Opaque

Brittle
Why Hash Representation is Opaque?

Recall that computations have

✓ Structure
✓ Semantics
✓ Analyzed
✓ Verified
Why Hash Representation is Opaque?

Recall that computations have

✓ Structure
✓ Semantics
✓ Analyzed
✓ Verified
Why Hash Representation is Opaque?

Recall that computations have

✓ Structure
✓ Semantics
✓ Analyzed
✓ Verified

Trust Policy: Homer trusts $\text{Hash}(\{P\})$ if $\{P\}$ is secure
Why Hash Representation is Opaque?

Recall that computations have
✓ Structure
✓ Semantics
✓ Analyzed
✓ Verified

**Trust Policy:** Homer trusts $\text{Hash}\{P\}$ if $\{P\}$ is secure
Why Hash Representation is Opaque?

Recall that computations have
✓ Structure
✓ Semantics
✓ Analyzed
✓ Verified

Hash representation loses
✘ Structure
✘ Semantics
✘ Analyzed
✘ Verified
Why Hash Representation is Brittle?

Recall that computations have

✓ Structure
✓ Semantics
✓ Analyzed
✓ Verified

No equational reasoning between computation principals

\[
\frac{1}{2} = \frac{2}{4}
\]
Why Hash Representation is Brittle?

{P} → Semantics-preserving compilation → {P'}
Why Hash Representation is Brittle?

No equational reasoning

- $P \approx P' \not\Rightarrow \text{Hash}(P) = \text{Hash}(P')$

- Equivalent programs are treated as different principals

- Whenever the computation changes, trust policy changes
Coal addresses both the challenges
Coal addresses both the challenges

Computation principals can be analyzed for intensional properties
Coal addresses both the challenges

Computation principals can be analyzed for intensional properties

Equivalent computations are treated as equivalent principals

$\frac{1}{2} = \frac{2}{4}$
Overview

e ::= ... | \mu T.e | \text{exec}(e)

\tau ::= ... | p \text{ says } \tau | \text{code}\{\mu T.e\}
Overview

ML/DCC-like

e ::= ... | μT.e | exec(e)

τ ::= ... | p says τ | code{μT.e}
Overview

ML/DCC-like

\[ e ::= \ldots \mid \mu T.e \mid \text{exec}(e) \]

\[ \tau ::= \ldots \mid p \text{ says } \tau \mid \text{code}\{\mu T.e\} \]

Principal \( p \) supports proposition \( \tau \)
Overview

ML/DCC-like

\[ e ::= \ldots \mid \mu T.e \mid \text{exec}(e) \]

Computation Expression

\[ \tau ::= \ldots \mid p \text{ says } \tau \mid \text{code}\{\mu T.e\} \]

Principal \(p\) supports proposition \(\tau\)
Overview

ML/DCC-like

$e ::= \ldots \mid \mu T.e \mid \text{exec}(e)$

$\tau ::= \ldots \mid p \text{ says } \tau \mid \text{code}\{
\mu T.e\}$

Principal $p$ supports proposition $\tau$

Computation Principal

Computation Expression
Overview

ML/DCC-like

Computation Expression

\[
e ::= \ldots \mid \mu T.e \mid \text{exec}(e)
\]

\[
\tau ::= \ldots \mid p \text{ says } \tau \mid \text{code}\{\mu T.e\}
\]

Principal \( p \) supports proposition \( \tau \)

Run a Computation Principal

Computation Principal
Assume \( \{P\} = \mu T.e \)

Homer trusts \( \{P\} \)
Specifying Trust in a Computation

∀X. code{μT.e} says X → Homer says X

Assume \{P\} = μT.e

Atomic Principal

Computation Principal
Chain of Trust
Specifying Chain of Trust

Homer trusts $\{P\}$ that is analyzed to be differentially private.
Homer trusts $\{P\}$ that is analyzed to be differentially private
Specifying Trust Chain in a Computation

∀X. code{μT.e} says X → Homer says X

Atomic Principal

Computation Principal

Assume \{P\} = μT.e
Specifying Trust Chain in a Computation

Code{DPAnalyzer} says (isDP \( \mu T.e \)) \( \rightarrow \forall X. \) code{\( \mu T.e \)} says X \( \rightarrow \) Homer says X

Assume \( \{P\} = \mu T.e \)

Atomic Principal

Computation Principal
Specifying Trust Chain in a Computation

\[ \forall X. \text{code}\{\mu T.e\} \rightarrow \text{Homer says } X \]

Assume \( \{P\} = \mu T.e \)

Code\{DPAnalyzer\} says (isDP \( \mu T.e \)) \( \rightarrow \forall X. \text{code}\{\mu T.e\} \text{ says } X \) \( \rightarrow \) Homer says \( X \)
Specifying **Trust Chain** in a Computation

\[
\forall X. \text{code}\{\mu T. e\} \rightarrow \text{Homer says } X
\]

Assume \( \{P\} = \mu T. e \)

\[
\text{Code}\{\text{DPAnalyzer}\} \rightarrow (\text{isDP } \mu T. e) \rightarrow \forall X. \text{code}\{\mu T. e\} \rightarrow \text{Homer says } X
\]
Homer trusts $\{P\}$ that is analyzed to be differentially private by a verified (differential privacy) analyzer.
Specifying **Trust Chain** in a Computation

Assume \( \{P\} = \mu T.e \)

\[ \text{code\{DPAnalyzer\}} \text{ says (isDP } \mu T.e) \rightarrow \forall X. \text{ code\{\mu T.e\}} \text{ says } X \rightarrow \text{ Homer says } X \]
Specifying **Trust Chain** in a Computation

Coq says (✅ DPAnalyzer)

$$\text{code\{DPAnalyzer\}} \text{ says } (\text{isDP } \mu T.e) \rightarrow \forall X. \text{code\{\mu T.e\}} \text{ says } X \rightarrow \text{Homer says } X$$

Assume $\{P\} = \mu T.e$
Specifying Trust in Equivalent Computations

Assume \( \{P\} = \mu T.e \)

Homer trusts \( \{P\} \) that is analyzed to be differentially private
Specifying Trust in Equivalent Computations

Assume $\{P\} = \mu T.e$

How to specify that Homer trusts compiled $\{P\}$?
Specifying Trust in Equivalent Computations

Assume $\{P\} = \mu T.e$

How to specify that Homer trusts compiled $\{P\}$?
Type System, Briefly

Key features are to ensure that

✓ Computation principals are well-formed
✓ Proofs and computations are separate
  • Mixing proofs and computations is meaningless
✓ Decidable type inference
✓ Equivalent programs are treated as equivalent computation principals
Equivalent Computations

\[ \Gamma \vdash e_1 \equiv e_2 \]

\[ \Gamma \vdash \text{code}\{e_1\} \equiv \text{code}\{e_2\} \]
Equivalent Computations

**Equivalent Programs**

\[ \Gamma \vdash e_1 \equiv e_2 \]

\[ \Gamma \vdash \text{code}\{e_1\} \equiv \text{code}\{e_2\} \]
Equivalent Computations

Equivalent Programs

\[ \Gamma \vdash e_1 \equiv e_2 \]

\[ \Gamma \vdash \text{code}\{e_1\} \equiv \text{code}\{e_2\} \]

Equivalent computations are treated as equivalent principals
Specifying Trust in Equivalent Computations

code{DPAnalyzer} says (isDP μT.e) → ∀X. code{μT.e} says X → Homer says X
Specifying Trust in Equivalent Computations

\[ \text{code\{DPAnalyzer\}} \text{ says (isDP } \mu T.e) \rightarrow \forall X. \text{code\{\mu T.e\}} \text{ says } X \rightarrow \text{Homer says } X \]
Secure Compilation

\[ \mu T.e \equiv e' \]

code\{DPAnalyzer\} says (isDP \mu T.e) \rightarrow \forall X. \text{code\{\mu T.e\}} says X \rightarrow \text{Homer says X}
Specifying Trust in Equivalent Computations

- **Secure Compilation**
  \[ \mu T.e = e' \Rightarrow \mu T.e = e' \]

- **Equivalent Principals**
  \[ \text{code}\{\mu T.e\} = \text{code}\{e'\} \]

- code\{DPAnalyzer\} says (isDP \(\mu T.e\)) \(\forall X.\text{code}\{\mu T.e\} \text{ says } X \Rightarrow \text{Homer says } X\)
Specifying Trust in Equivalent Computations

**Secure Compilation**

\[
\mu T.e = e' \implies \mu T.e \equiv e'
\]

\[
\text{code}\{\mu T.e\} \equiv \text{code}\{e'\}
\]

**Equivalent Principals**

\[
\forall X. \text{code}\{\mu T.e\} \text{ says } X \implies \text{Homer says } X
\]

\[
\equiv
\]

\[
\forall X. \text{code}\{e'\} \text{ says } X \implies \text{Homer says } X
\]
Specifying Trust in Equivalent Computations

**Secure Compilation**

\[ \mu T.e = e' \Rightarrow \mu T.e \equiv e' \]

\[ \text{code}\{\mu T.e\} \equiv \text{code}\{e'\} \]

**Equivalent Principals**

\[
\text{code}\{\text{DPAnalyzer}\} \text{ says (isDP } \mu T.e) \rightarrow \forall X. \text{ code}\{\mu T.e\} \text{ says } X \rightarrow \text{ Homer says } X
\]

\[ \equiv \]

\[
\text{code}\{\text{DPAnalyzer}\} \text{ says (isDP } \mu T.e) \rightarrow \forall X. \text{ code}\{e'\} \text{ says } X \rightarrow \text{ Homer says } X
\]
Coal: Next Steps
Towards a real language: A secure programming language based on Coal

- Realize Coal abstractions (e.g., Intel SGX as a computation principal)
- Information-flow control guarantees
  - E.g. strong integrity guarantees for computation principals (they do not err)
Coal: Next Steps

❖ Towards a real language: A secure programming language based on Coal
  • Realize Coal abstractions (e.g., Intel SGX as a computation principal)
  • Information-flow control guarantees
    • E.g. strong integrity guarantees for computation principals (they do not err)
❖ Explore various notions of program equivalence to get equivalent principals
  • Introduces functional dependent types
  • Type checking could be undecidable
Coal: Enables expressive authorization policies using computation principals
Backup
Case Study: eBPF Authorization
Kernel says \( \forall U. \text{ Verifier says (terminates } U \land \text{ safeSysCalls } U \) \) \( \rightarrow \) \( U \Rightarrow \text{ Kernel} \)
Specifying Trust Chain in Equivalent Computations

Coq says (✅ DPAnalyzer)

code{DPAnalyzer} says (isDP μT.e) → ∀X. code{μT.e} says X → Homer says X
Specifying Trust Chain in Equivalent Computations

Equivalent Principals

Coq says \((\checkmark \text{ DPAnalyzer})\)

\[
\text{code}\{\text{DPAnalyzer}\} \equiv \text{code}\{\text{DPAnalyzer}\}
\]

Homer says \(X\)
Specifying **Trust Chain** in Equivalent Computations

Equivalent Principals

Coq says (✅ DPAnalyzer)

\[ \text{code}\{\text{DPAnalyzer}\} \equiv \text{code}\{ \text{〖DPAnalyzer〗} \} \]

Homer says X

\[ \forall X. \text{code}\{\mu T.e\} \text{ says } X \Rightarrow \text{Homer says } X \]

Coq says (✅ DPAnalyzer)

\[ \text{code}\{\text{〖DPAnalyzer〗} \} \text{ says (isDP } \mu T.e) \Rightarrow \forall X. \text{code}\{\mu T.e\} \text{ says } X \Rightarrow \text{Homer says } X \]