Expressive Authorization Policies using Computation Principals

Anitha Gollamudi and Stephen Chong
Harvard University
Goal

Express Authorization Policies involving Computations
Pay $42 to IRS
CODE

Pay $42 to IRS

Pay $42 to IRS
CODE speaks for
CODE

is a

Computation Principal
Examples

- Hash Digest
- Mobile code
- Smart Contracts
- Trusted Execution Environments
Trusted Execution Environment (TEE)

- TEE provides confidentiality and integrity to the code running inside
  - e.g. Intel SGX, ARM TrustZone
- Identity of TEE = identity of the code
- Offers remote code attestation

*TEE enables delegating trust to a computation independent of the machine on which it is executed*
Simple Policy

CODE

speaks for
Complex Policy

Arbitrary code verified by a proof assistant
Contribution

An authorization logic with a mechanism that elegantly expresses trust directly to the code using *Computation Principals*
Our authorization logic extends Abadi’s Polymorphic Dependency Core Calculus (DCC)
Polymorphic DCC

- A calculus and authorization logic for access control
- Extends Moggi’s computational lambda calculus
  - \((\eta_A \ e)\) protects \(e\) at level \(A\)
    - has the type “\(A\) says …”
  - \(\text{bind } x = (\eta_A \ e) \text{ in } e'\) privileged operation authorized by \(A\)
- Polymorphism
Encoding “A speaks for B”
Encoding “A speaks for B”

Type (Proposition)

∀X. A says X → B says X
Encoding "A speaks for B"

Type (Proposition)

\( \forall X. \text{A says } X \rightarrow \text{B says } X \)

Term (Proof)

\( \Lambda X. \lambda x: \text{A says } X. \text{bind } y = x \text{ in } (\eta_B y) \)

Curry-Howard Correspondence
C-DCC

e ::= ... | μt.e | exec(e, e')

t ::= ... | CP μt.e
C-DCC

Computation Expression

e ::= ... | μt.e | exec(e, e')

τ ::= ... | CP μt.e
C-DCC

Computation Expression

\[ e ::= … | \mu t.e | \text{exec}(e, e') \]

Computation Principal

\[ \tau ::= … | \text{CP} \mu t.e \]
C-DCC

Computation Principal

Computation Expression

\[ e ::= \ldots \mid \mu t.e \mid \text{exec}(e, e') \]

\[ \tau ::= \ldots \mid \text{CP} \mu t.e \]

Run computation
∀X. (CP μt. < $ > ) says X → A says X

Dependent Type
∀P::Computation Principals. (Verified P) → ∀X. P says X → A says X

Predicates on Computation Principals