Relating Cryptography and Polymorphism

Eijiro Sumii
Joint Work with Benjamin Pierce
University of Pennsylvania
Main Result

- Adaptation of relational parametricity from polymorphic $\lambda$-calculus to cryptographic $\lambda$-calculus

\[ e \sim e' : \tau \Rightarrow e \approx e' \]

- useful for reasoning about programs using encryption
  - E.g., (in)correctness proof of security protocols
Outline

- Background
- Parametricity for Type Abstraction
- Cryptographic $\lambda$-Calculus
- Parametricity for Encryption
- Current Status and Future Work
Two Approaches to Information Hiding

♦ Type abstraction
  - conceals the types of data
    • existential types, universal types, modules, packages, etc.

♦ Encryption
  - obfuscates the values of data
Example

- Using type abstraction:
  \[
  \text{pack int, } \langle 3, \lambda x. \ x \mod 2 \rangle \\
  \text{as } \square \alpha. \alpha \times (\alpha \rightarrow \text{int})
  \]

- Using encryption:
  \[
  \text{new k in } \langle \{3\}_k, \lambda \{x\}_k. \ x \mod 2 \rangle \\
  \text{new k in } ...
  \]
  generate a fresh key \( k \)
  \[
  \{v\}_k
  \]
  a value \( v \) encrypted by the key \( k \)
Secrecy as Non-Interference as Contextual Equivalence

♦ Relational parametricity [Reynolds 83] (or representation independence [Mitchell 86]):

\[
\text{pack int, } \langle 3, \lambda x. x \mod 2 \rangle \\
\quad \text{as } \square \alpha. \alpha \times (\alpha \to \text{int}) \\
\approx \text{pack int, } \langle 1, \lambda x. x \mod 2 \rangle \\
\quad \text{as } \square \alpha. \alpha \times (\alpha \to \text{int})
\]

♦ This work:

\[
\text{new k in } \langle \{3\}_k, \lambda \{x\}_k. x \mod 2 \rangle \\
\approx \text{new k in } \langle \{1\}_k, \lambda \{x\}_k. x \mod 2 \rangle
\]
Outline

- Background
- Parametricity for Type Abstraction
- Cryptographic $\lambda$-Calculus
- Parametricity for Encryption
- Current Status and Future Work
Principle of Parametricity: "Related" Values are Equivalent

\[ \varphi \models e \sim e' : \tau \text{ for some } \varphi \]

\[ \Downarrow \]

\[ \varnothing f e \varnothing = \varnothing f e' \varnothing \text{ for any } f : \tau \rightarrow bool \]

- \( \sim \) is defined by induction on \( \tau \)
- \( \varphi \) defines the case of abstract types, mapping each free type variable to a relation between values of its concrete types
Definition of the Logical Relation (1/2)

- Values of a base type (e.g. int) are related iff they are equal
- Functions are related iff they map related arguments to related results
- Pairs are related iff their elements are respectively related
Definition of the Logical Relation (2/2)

♦ Packages are related iff their implementations can be related

\[ \phi \quad \text{pack } \tau, v \quad \text{as } \quad \square \alpha. \sigma \sim \]
\[ \text{pack } \tau', v' \quad \text{as } \quad \square \alpha. \sigma : \quad \square \alpha. \sigma \iff \]
\[ \phi, \alpha \mapsto r \quad \square \quad v[\tau/\alpha] \sim v'[\tau'/\alpha] \quad : \quad \sigma \]
for some \( r \subseteq \tau \times \tau' \)

♦ Values of an abstract type \( \alpha \) are related iff they are related by \( \phi(\alpha) \)

\[ \phi \quad v \sim v' : \quad \alpha \iff (v, v') \in \phi(\alpha) \]
Example

\[ \alpha \mapsto \{(3,1)\} \]

\[ \langle 3, \lambda x. \text{x mod 2} \rangle \sim \langle 1, \lambda x. \text{x mod 2} \rangle \]

: \( \alpha \times (\alpha \to \text{int}) \)

\[ \Downarrow \]

\[ \sqcap \text{pack int, } \langle 3, \lambda x. \text{x mod 2} \rangle \]

as \[ \sqcap \alpha. \alpha \times (\alpha \to \text{int}) \]

\[ \sim \text{pack int, } \langle 1, \lambda x. \text{x mod 2} \rangle \]

as \[ \sqcap \alpha. \alpha \times (\alpha \to \text{int}) : \sqcap \alpha. \alpha \times (\alpha \to \text{int}) \]
Outline

♦ Background
♦ Parametricity for Type Abstraction
♦ Cryptographic $\lambda$-Calculus
♦ Parametricity for Encryption
♦ Current Status and Future Work
Cryptographic $\lambda$-Calculus: Syntax and Semantics

Simply-typed call-by-value $\lambda$-calculus + shared-key cryptographic primitives

$\text{let } \{x\}_{e_1} = e_2 \text{ in } e_3 \text{ else } e_4$

$\text{let } \{x\}_k = \{v\}_{k'} \text{ in } e \text{ else } e' \rightarrow e[v/x] \text{ if } k = k', e' \text{ otherwise}$
Cryptographic \( \lambda \)-Calculus: Typing Rules

\[ \Gamma \vdash k : \text{key} \]

\[ \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \text{key}}{\Gamma \vdash \{e_1\}_e^2 : \text{bits}(\tau)} \]

\[ \frac{\Gamma \vdash e_1 : \text{key} \quad \Gamma \vdash e_2 : \text{bits}(\tau')} \quad \Gamma, x : \tau' \vdash e_3 : \tau \quad \Gamma \vdash e_4 : \tau}{\Gamma \vdash \text{let } \{x\}_{e_1} = e_2 \text{ in } e_3 \text{ else } e_4 : \tau} \]
Outline

♦ Background
♦ Parametricity for Type Abstraction
♦ Cryptographic \( \lambda \)-Calculus
♦ Parametricity for Encryption
♦ Current Status and Future Work
Parametricity Adapted

\[ \phi \triangleq e \sim e' : \tau \quad \text{for some } \varphi \]

\[ \Downarrow \]

\[ \Box f e \Box = \Box f e' \Box \quad \text{for any } f : \tau \to \text{bool} \]

s.t. \( \text{dom}(\varphi) \cap \text{keys}(f) = \emptyset \)

\(~\) is defined by induction on \(\tau\), meaning that \(e\) and \(e'\) are equivalent and \text{don't leak the secret keys}.

\(\varphi\) defines the case of \(\text{bits}(\tau)\), mapping each secret key to a relation between values encrypted by the key.
Definition of the Logical Relation

- Keys are related iff they are equal and non-secret
  \[ \varphi \sqcap k \sim k : \text{key} \iff k \notin \text{dom}(\varphi) \]

- Values encrypted by a secret key \( k \) are related iff they are related by \( \varphi(k) \)
  \[ \varphi \sqcap \{v\}_k \sim \{v'\}_k : \text{bits}(\tau) \iff (v,v') \in \varphi(k) \text{ if } k \in \text{dom}(\varphi) \]
  \[ \varphi \sqcap v \sim v' : \tau \text{ if } k \notin \text{dom}(\varphi) \]
Example

\[ k \mapsto \{(3,1)\} \quad \{3\}_k \sim \{1\}_k : \text{bits(int)} \]

\[ k \mapsto \{(3,1)\} \quad \lambda \{x\}_k \cdot x \mod 2 \]
\[ \sim \lambda \{x\}_k \cdot x \mod 2 : \text{bits(int)} \rightarrow \text{int} \]
\[ \Downarrow \]

\[ k \mapsto \{(3,1)\} \quad \langle \{3\}_k, \lambda \{x\}_k \cdot x \mod 2 \rangle \sim \langle \{1\}_k, \lambda \{x\}_k \cdot x \mod 2 \rangle \]
\[ : \text{bits(int)} \times (\text{bits(int)} \rightarrow \text{int}) \]

N.B.

\[ \lambda \{x\}_k \cdot e \equiv \lambda z. \text{let } \{x\}_k = z \text{ in } e \text{ else } \bot \]
Outline

♦ Background
♦ Parametricity for Type Abstraction
♦ Cryptographic $\lambda$-Calculus
♦ Parametricity for Encryption
♦ Current Status and Future Work
Current Status

♦ Treatment of fresh key generation, adapting [Stark 97]
♦ (In)correctness proof of a few security protocols, using the following encoding
  • principal = function from messages to messages (with its own continuation)
  • configuration = record of principals and non-secret keys
  • network and scheduler = "right" context
  • attacker = arbitrary context
Fresh Key Generation (1/2)

- **Syntax**

\[ e ::= \ldots \mid \text{new } x \text{ in } e \]

- **Semantics**

\[ e \downarrow (S)v \]

read as: "the expression e evaluates to the value v, generating the set S of fresh keys"; note that (S) is a binder
Fresh Key Generation (2/2)

- Logical relation

\[ \varphi \triangleq e \sim e' : \tau \iff \]
\[ e \downarrow (\{k_1, \ldots, k_n\} \oplus S) \trianglerighteq v_1, \]
\[ e' \downarrow (\{k_1, \ldots, k_n\} \oplus S') \trianglerighteq v_2, \]
and
\[ \varphi, k_1 \mapsto r_1, \ldots, k_n \mapsto r_n \triangleq v_1 \sim v_2 : \tau \]
for some \( k_1, \ldots, k_n, r_1, \ldots, r_n, S \) and \( S' \)

For example,

- new k in \( \langle \{3\}_k, \lambda \{x\}_k. x \text{ mod } 2 \rangle \sim \)
new k in \( \langle \{1\}_k, \lambda \{x\}_k. x \text{ mod } 2 \rangle \)
\( : \text{bits(int)} \times (\text{bits(int)} \rightarrow \text{int}) \)
Future Work

- Recursive functions/types
  cf. [Birkedal & Harper 97], [Crary & Harper], etc.

- Concurrency and distribution
  cf. spi-calculus [Abadi & Gordon 97],
  evaluation semantics for CCS [Pitts 96],
  typed equivalence in polymorphic
  \(\pi\)-calculus [Pierce & Sangiorgi 97],
  parametricity in linear polymorphic
  \(\lambda\)-calculus [Pitts 2000], etc.