Iris: Higher-Order Concurrent Separation Logic

Lecture 22: Asynchronous reasoning about randomized programs

Philipp G. Haselwarter

Aarhus University, Denmark

December 7, 2023

Overview

Earlier:

- Operational Semantics of $\lambda_{\mathrm{ref,conc}}$: e, $(h, e) \rightsquigarrow (h, e')$, and $(h, \mathcal{E}) \to (h', \mathcal{E}')$
- Basic Logic of Resources : $l \hookrightarrow v$, $P * Q$, $P \mathrel{-\!\!*} Q$, $\Gamma \mid P \vdash Q$
- Basic Separation Logic : $\{P\}\, e\, \{v.Q\}$: Prop, isList l xs, ADTs, foldr
- Later ($\triangleright$) and Persistent ($\square$) Modalities.
- Concurrency Intro, Invariants and Ghost State
- CAS, Spin Locks, Concurrent Counter Modules.
- Monotone Resource Algebra
- Case studies: Ticket Lock, Array Based Queuing Lock, and Stack with Helping
- More details of constructions, *e.g.*, weakest preconditions, *etc.*
- Logical Relations for safety & type abstraction in Iris
- Randomization, $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ operational semantics, contextual & logical refinement

Today:

- Case study: Security of ElGamal public key encryption
- Lazy vs eager sampling
- Asynchronous coupling rules
- Ongoing & future work

The ElGamal public key scheme

$$
\begin{aligned}
keygen &\triangleq \lambda\,\_\,.\ \mathsf{let}\ sk := \mathsf{rand}(n)\ \mathsf{in} \\
&\qquad\quad \mathsf{let}\ pk := g^{sk}\ \mathsf{in} \\
&\qquad\quad (sk, pk) \\
dec &\triangleq \lambda\ sk\ (B, X).\ \ X \cdot B^{-sk}
\end{aligned}
\qquad
\begin{aligned}
enc &\triangleq \lambda\ pk\ \ msg.\ \mathsf{let}\ b := \mathsf{rand}(n)\ \mathsf{in} \\
&\qquad\qquad\quad \mathsf{let}\ B := g^{b}\ \mathsf{in} \\
&\qquad\qquad\quad \mathsf{let}\ X := msg \cdot pk^{b}\ \mathsf{in} \\
&\qquad\qquad\quad (B, X)
\end{aligned}
$$

The ElGamal public key scheme

$$
\begin{aligned}
keygen &\triangleq \lambda\, \_\,.\ \mathsf{let}\ sk := \mathsf{rand}(n)\ \mathsf{in} \\
&\qquad\quad \mathsf{let}\ pk := g^{sk}\ \mathsf{in} \\
&\qquad\quad (sk, pk) \\
dec &\triangleq \lambda\ sk\ (B, X).\ \ X \cdot B^{-sk}
\end{aligned}
\qquad
\begin{aligned}
enc &\triangleq \lambda\ pk\ \ msg.\ \mathsf{let}\ b := \mathsf{rand}(n)\ \mathsf{in} \\
&\qquad\qquad\quad \mathsf{let}\ B := g^{b}\ \mathsf{in} \\
&\qquad\qquad\quad \mathsf{let}\ X := msg \cdot pk^{b}\ \mathsf{in} \\
&\qquad\qquad\quad (B, X)
\end{aligned}
$$

Parameterized by a group G encoding messages, ciphertexts, and keys.

Write $G = (1, \cdot, -^{-1})$ for a finite cyclic group of order $|G|$, generated by g, and let $n = |G| - 1$.

Public key security, I

$keygen \triangleq \lambda\,\_\,.$
$\quad \text{let } sk := \text{rand}(n) \text{ in}$
$\quad \text{let } pk := g^{sk} \text{ in}$
$\quad (sk, pk)$

$enc \triangleq \lambda\ pk\ \ msg.$
$\quad \text{let } b := \text{rand}(n) \text{ in}$
$\quad \text{let } B := g^{b} \text{ in}$
$\quad \text{let } X := msg \cdot pk^{b} \text{ in}$
$\quad (B, X)$

$PK_{real} \triangleq$
$\text{let } (sk, pk) := keygen() \text{ in}$
$\text{let } count := \text{ref } 0 \text{ in}$
$\text{let } query = \lambda\ msg.$
$\quad \text{if } !\,count \neq 0 \text{ then}$
$\quad\quad \text{None}$
$\quad \text{else}$
$\quad\quad count \leftarrow 1;$
$\quad\quad \text{let } (B, X) = enc\ \ pk\ \ msg \text{ in}$
$\quad\quad \text{Some } (B, X)$
$\text{in } (pk, query)$

$PK_{rand} \triangleq$
$\text{let } (sk, pk) := keygen() \text{ in}$
$\text{let } count := \text{ref } 0 \text{ in}$
$\text{let } query = \lambda\ msg.$
$\quad \text{if } !\,count \neq 0 \text{ then}$
$\quad\quad \text{None}$
$\quad \text{else}$
$\quad\quad count \leftarrow 1;$
$\quad\quad \text{let } b := \text{rand}(n) \text{ in}$
$\quad\quad \text{let } x := \text{rand}(n) \text{ in}$
$\quad\quad \text{let } (B, X) := (g^{b}, g^{x}) \text{ in}$
$\quad\quad \text{Some } (B, X)$
$\text{in } (pk, query)$

The Decisional Diffie-Hellman assumption

Assumption: DH_{real} and DH_{rand} are hard to distinguish.

$$DH_{real} \triangleq \begin{array}{l} \text{let } a := \text{rand}(n) \text{ in} \\ \text{let } b := \text{rand}(n) \text{ in} \\ (g^a, g^b, g^{ab}) \end{array} \qquad DH_{rand} \triangleq \begin{array}{l} \text{let } a := \text{rand}(n) \text{ in} \\ \text{let } b := \text{rand}(n) \text{ in} \\ \text{let } c := \text{rand}(n) \text{ in} \\ (g^a, g^b, g^c) \end{array}$$

Proof idea: exhibit $\mathcal{C}$ s.t. we can prove

$$\vdash PK_{real} \simeq_{\text{ctx}} \mathcal{C}[DH_{real}] : \tau_{PK} \tag{1}$$

$$\vdash PK_{rand} \simeq_{\text{ctx}} \mathcal{C}[DH_{rand}] : \tau_{PK} \tag{2}$$

Then PK_{real} and PK_{rand} should also be hard to distinguish by our assumption.

Public key security, II

$$
\begin{array}{ll}
PK_{real} \triangleq & PK_{rand} \triangleq \\
\text{let } (sk, pk) := keygen() \text{ in} & \text{let } (sk, pk) := keygen() \text{ in} \\
\text{let } count := \text{ref } 0 \text{ in} & \text{let } count := \text{ref } 0 \text{ in} \\
\text{let } query = \lambda\ msg. & \text{let } query = \lambda\ msg. \\
\quad \text{if } !\,count \neq 0 \text{ then} & \quad \text{if } !\,count \neq 0 \text{ then} \\
\qquad \text{None} & \qquad \text{None} \\
\quad \text{else} & \quad \text{else} \\
\qquad count \leftarrow 1; & \qquad count \leftarrow 1; \\
 & \qquad \text{let } b := \text{rand}(n) \text{ in} \\
 & \qquad \text{let } x := \text{rand}(n) \text{ in} \\
\qquad \text{let } (B, X) = enc\ pk\ msg \text{ in} & \qquad \text{let } (B, X) := (g^b, g^x) \text{ in} \\
\qquad \text{Some } (B, X) & \qquad \text{Some } (B, X) \\
\text{in } (pk, query) & \text{in } (pk, query)
\end{array}
$$

(a) The security games.

Public key security, II

$$
\begin{array}{l}
PK_{real} \triangleq \\
\text{let } (sk, pk) := keygen() \text{ in} \\
\text{let } count := \text{ref } 0 \text{ in} \\
\text{let } query = \lambda\ msg. \\
\quad \text{if } !\,count \neq 0 \text{ then} \\
\quad\quad \text{None} \\
\quad \text{else} \\
\quad\quad count \leftarrow 1; \\
\quad\quad \text{let } (B, X) = enc\ pk\ msg \text{ in} \\
\quad\quad \text{Some } (B, X) \\
\text{in } (pk, query)
\end{array}
\qquad
\begin{array}{l}
PK_{rand} \triangleq \\
\text{let } (sk, pk) := keygen() \text{ in} \\
\text{let } count := \text{ref } 0 \text{ in} \\
\text{let } query = \lambda\ msg. \\
\quad \text{if } !\,count \neq 0 \text{ then} \\
\quad\quad \text{None} \\
\quad \text{else} \\
\quad\quad count \leftarrow 1; \\
\quad\quad \text{let } b := \text{rand}(n) \text{ in} \\
\quad\quad \text{let } x := \text{rand}(n) \text{ in} \\
\quad\quad \text{let } (B, X) := (g^b, g^x) \text{ in} \\
\quad\quad \text{Some } (B, X) \\
\text{in } (pk, query)
\end{array}
$$

(a) The security games.

$$
\begin{array}{l}
\mathcal{C}[-] \triangleq \\
\text{let } (pk, B, C) := - \text{ in} \\
\text{let } count := \text{ref } 0 \text{ in} \\
\text{let } query = \lambda\ msg. \\
\quad \text{if } !\,count \neq 0 \text{ then} \\
\quad\quad \text{None} \\
\quad \text{else} \\
\quad\quad count \leftarrow 1; \\
\quad\quad \text{let } X = msg \cdot C \text{ in} \\
\quad\quad \text{Some } (B, X) \\
\text{in } (pk, query)
\end{array}
$$

(b) The DH reduction context.

ElGamal security reduction, I

$$PK_{real} \qquad \simeq_{\text{ctx}} \qquad \mathcal{C}[DH_{real}]$$

PK_{real}	$\mathcal{C}[DH_{real}]$
	let $(pk, B, C) =$
let $sk := \text{rand}(n)$ in	let $a := \text{rand}(n)$ in
let $pk := g^{sk}$ in	let $b := \text{rand}(n)$ in
	(g^a, g^b, g^{ab}) in
let $count := \text{ref } 0$ in	let $count := \text{ref } 0$ in
let $query = \lambda\ msg.$	let $query = \lambda\ msg.$
if $!\,count \neq 0$ then	if $!\,count \neq 0$ then
None	None
else	else
$count \leftarrow 1;$	$count \leftarrow 1;$
let $b = \text{rand}(n)$ in	
let $B = g^b$ in	
let $X = msg \cdot pk^b$ in	let $X := msg \cdot C$ in
Some (B, X)	Some (B, X)
in $(pk, query)$	in $(pk, query)$

Isolating the problem: lazy/eager sampling

$$eager \triangleq \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in}\ \lambda\_.\ b$$

$$\begin{aligned}
lazy \triangleq\ & \mathsf{let}\ r := \mathsf{ref}(\mathsf{None})\ \mathsf{in} \\
& \quad \lambda_.\ \mathsf{match}\ !\,r\ \mathsf{with} \\
& \qquad \mathsf{Some}(b) \Rightarrow b \\
& \qquad \mid \mathsf{None} \quad \Rightarrow \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in} \\
& \qquad \qquad \qquad \quad r \leftarrow \mathsf{Some}(b); \\
& \qquad \qquad \qquad \quad b \\
& \qquad \mathsf{end}
\end{aligned}$$

Isolating the problem: lazy/eager sampling

$$\mathit{eager} \triangleq \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in}\ \lambda\_.\ b$$

$$\begin{aligned}
\mathit{lazy} \triangleq\ & \mathsf{let}\ r := \mathsf{ref}(\mathrm{None})\ \mathsf{in} \\
& \quad \lambda_.\ \mathsf{match}\ !r\ \mathsf{with} \\
& \qquad \mathrm{Some}(b) \Rightarrow b \\
& \qquad \mid \mathrm{None} \quad \Rightarrow \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in} \\
& \qquad\qquad\qquad\quad r \leftarrow \mathrm{Some}(b); \\
& \qquad\qquad\qquad\quad b \\
& \quad \mathsf{end}
\end{aligned}$$

We expect

$$\vdash \mathit{lazy} \simeq_{\mathrm{ctx}} \mathit{eager} : \mathrm{unit} \to \mathrm{bool}$$

Recall: we plug *lazy*, *eager* into a well-typed context evaluating to a boolean.

Not the same distribution on values, but same observations!

Isolating the problem: lazy/eager sampling

$$eager \triangleq \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in}\ \lambda\_.\ b$$

$$\begin{aligned} lazy \triangleq\ & \mathsf{let}\ r := \mathsf{ref}(\mathrm{None})\ \mathsf{in} \\ & \lambda\_.\ \mathsf{match}\ !\,r\ \mathsf{with} \\ & \quad \mathrm{Some}(b) \Rightarrow b \\ & \quad \mid \mathrm{None} \Rightarrow \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in} \\ & \qquad\qquad r \leftarrow \mathrm{Some}(b); \\ & \qquad\qquad b \\ & \quad \mathsf{end} \end{aligned}$$

We expect

$$\vdash lazy \simeq_{\mathrm{ctx}} eager : \mathrm{unit} \to \mathrm{bool}$$

$$\textsc{rel-couple-rands}\ \frac{f \text{ bijection} \qquad \forall n \leq N.\, \Delta \vDash_{\mathcal{E}} E[n] \precsim E'[f(n)] : \tau}{\Delta \vDash_{\mathcal{E}} E[\mathsf{rand}(N)] \precsim E'[\mathsf{rand}(N)] : \tau}$$

Isolating the problem: lazy/eager sampling

$$eager \triangleq \text{let } b := \text{flip}() \text{ in } \lambda\_.\ b$$

$$\begin{aligned} lazy \triangleq\ & \text{let } r := \text{ref}(\text{None}) \text{ in} \\ & \quad \lambda\_.\ \text{match } !r \text{ with} \\ & \qquad \text{Some}(b) \Rightarrow b \\ & \qquad \mid \text{None} \Rightarrow \text{let } b := \text{flip}() \text{ in} \\ & \qquad\qquad\qquad r \leftarrow \text{Some}(b); \\ & \qquad\qquad\qquad b \\ & \quad \text{end} \end{aligned}$$

We expect

$$\vdash lazy \simeq_{\text{ctx}} eager : \text{unit} \to \text{bool}$$

REL-COUPLE-RANDS

$$\frac{f \text{ bijection} \qquad \forall n \le N.\ \Delta \vDash_{\mathcal{E}} E[n] \precsim E'[f(n)] : \tau}{\Delta \vDash_{\mathcal{E}} E[\text{rand}(N)] \precsim E'[\text{rand}(N)] : \tau}$$

Does not apply.

Isolating the problem: lazy/eager sampling

$$eager \triangleq \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in}\ \lambda\_.\ b$$

$$\begin{aligned}
lazy \triangleq\ & \mathsf{let}\ r := \mathsf{ref}(\mathsf{None})\ \mathsf{in} \\
& \quad \lambda_.\ \mathsf{match}\ !\,r\ \mathsf{with} \\
& \qquad \mathsf{Some}(b) \Rightarrow b \\
& \qquad \mid \mathsf{None} \Rightarrow \mathsf{let}\ b := \mathsf{flip}()\ \mathsf{in} \\
& \qquad\qquad\qquad r \leftarrow \mathsf{Some}(b); \\
& \qquad\qquad\qquad b \\
& \quad \mathsf{end}
\end{aligned}$$

We expect

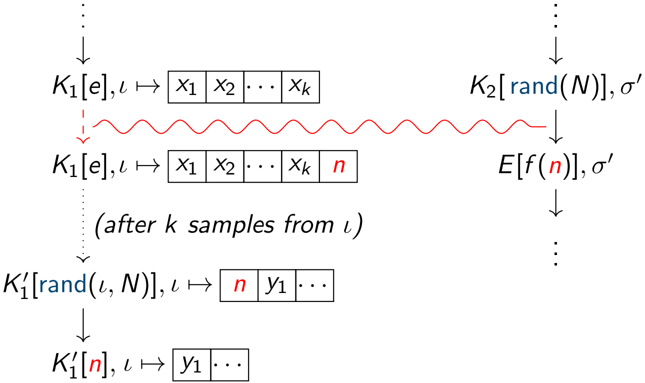
$$\vdash lazy \simeq_{\mathsf{ctx}} eager : \mathsf{unit} \to \mathsf{bool}$$

REL-COUPLE-RANDS

$$\frac{f \text{ bijection} \qquad \forall n \leq N.\ \Delta \vDash_{\mathcal{E}} E[n] \precsim E'[f(n)] : \tau}{\Delta \vDash_{\mathcal{E}} E[\mathsf{rand}(N)] \precsim E'[\mathsf{rand}(N)] : \tau}$$

Does not apply. 😿

High-level idea: labelled sampling and presampling tapes



An asynchronous coupling established through the rule REL-COUPLE-TAPE-L.

Syntax

Modify $\mathbf{F}^{\mathsf{rand}}_{\mu,\mathsf{ref}}$ as follows

$$
\begin{array}{llll}
 & & \vdots & \\
\mathit{Val} & v & ::= & \ldots \mid \iota \in \mathit{Label} \\
\mathit{Exp} & e & ::= & \ldots \mid \mathsf{rand}(e_1, e_2) \mid \mathsf{tape}\, e \\
\mathit{ECtx} & E & ::= & \ldots \mid \mathsf{rand}(e, K) \mid \mathsf{rand}(K, v) \mid \mathsf{tape}(K) \\
 & & & \\
\mathit{Heap} & h & \in & \mathit{Loc} \xrightharpoonup{\text{fin}} \mathit{Val} \\
\mathit{TapeMap} & & = & \mathit{Label} \xrightharpoonup{\text{fin}} \mathit{Tape} \\
\mathit{State} & \sigma & \in & \mathit{Heap} \times \mathit{TapeMap} \\
\mathit{Tape} & t & \in & \{(N, \vec{n}) \mid N \in \mathbb{N} \wedge \vec{n} \in \mathbb{N}^*_{\leq N}\} \\
\mathit{Config} & \rho & ::= & (\sigma, e) \\
 & & & \\
\mathit{Type} & \tau & ::= & \ldots \mid \mathsf{tape}
\end{array}
$$

Operational semantics

$$
\begin{array}{llll}
\sigma, \mathsf{tape}(N) & \to^{1} & \sigma[\iota \mapsto (N, \varepsilon)],\ \iota & \text{if } \iota = \mathsf{fresh}(\sigma) \\
\sigma, \mathsf{rand}(N, \iota) & \to^{1/(N+1)} & \sigma,\ n & \text{if } \sigma(\iota) = (N, \varepsilon) \text{ and } n \leq N \\
\sigma, \mathsf{rand}(N, \iota) & \to^{1} & \sigma[\iota \mapsto (N, \vec{n})],\ n & \text{if } \sigma(\iota) = (N, n \cdot \vec{n})
\end{array}
$$

Operational semantics

$$
\begin{array}{llll}
\sigma, \mathsf{tape}(N) & \to^{1} & \sigma[\iota \mapsto (N, \varepsilon)],\ \iota & \text{if } \iota = \mathsf{fresh}(\sigma) \\
\sigma, \mathsf{rand}(N, \iota) & \to^{1/(N+1)} & \sigma,\ n & \text{if } \sigma(\iota) = (N, \varepsilon) \text{ and } n \leq N \\
\sigma, \mathsf{rand}(N, \iota) & \to^{1} & \sigma[\iota \mapsto (N, \vec{n})],\ n & \text{if } \sigma(\iota) = (N, n \cdot \vec{n})
\end{array}
$$

No primitives in the language add values to the tapes!

Presampling steps are *ghost operations* appearing only in the relational logic.

A resource algebra for tapes

Recall

$$Tape = \{(N, \vec{n}) \mid N \in \mathbb{N} \wedge \vec{n} \in \mathbb{N}^*_{\leq N}\}$$

and

$$TapeMap = Label \xrightarrow{\text{fin}} Tape$$

We can define a RA like for heaps.

A resource algebra for tapes

Recall

$$\mathit{Tape} = \{(N, \vec{n}) \mid N \in \mathbb{N} \land \vec{n} \in \mathbb{N}^*_{\leq N}\}$$

and

$$\mathit{TapeMap} = \mathit{Label} \xrightarrow{\text{fin}} \mathit{Tape}$$

We can define a RA like for heaps. Tape ownership:

$$\iota \hookrightarrow (N, \vec{n})$$

A resource algebra for tapes

Recall

$$\mathit{Tape} = \{(N, \vec{n}) \mid N \in \mathbb{N} \land \vec{n} \in \mathbb{N}^*_{\leq N}\}$$

and

$$\mathit{TapeMap} = \mathit{Label} \xrightarrow{\text{fin}} \mathit{Tape}$$

We can define a RA like for heaps. Tape ownership:

$$\iota \hookrightarrow (N, \vec{n})$$

Likewise for the right-hand side "spec" program:

$$\iota \hookrightarrow_{\mathsf{s}} (N, \vec{n})$$

Reasoning with tapes: one-sided rules

REL-ALLOC-TAPE-L
$$\frac{\forall \iota.\, \iota \hookrightarrow (N, \varepsilon) \twoheadrightarrow\!\!* \ \Delta \vDash E[\iota] \precsim e : \tau}{\Delta \vDash E[\mathsf{tape}(N)] \precsim e : \tau}$$

REL-ALLOC-TAPE-R
$$\frac{\forall \iota.\, \iota \hookrightarrow_{\mathsf{s}} (N, \varepsilon) \twoheadrightarrow\!\!* \ \Delta \vDash e \precsim E[\iota] : \tau}{\Delta \vDash e \precsim E[\mathsf{tape}(N)] : \tau}$$

REL-RAND-TAPE-L
$$\frac{\iota \hookrightarrow (N, n \cdot \vec{n}) \qquad \iota \hookrightarrow (N, \vec{n}) \twoheadrightarrow\!\!* \ \Delta \vDash_{\mathcal{E}} E[n] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} E[\mathsf{rand}(N, \iota)] \precsim e_2 : \tau}$$

REL-RAND-TAPE-R
$$\frac{\iota \hookrightarrow_{\mathsf{s}} (N, n \cdot \vec{n}) \qquad \iota \hookrightarrow_{\mathsf{s}} (N, \vec{n}) \twoheadrightarrow\!\!* \ \Delta \vDash_{\mathcal{E}} e_1 \precsim E[n] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[\mathsf{rand}(N, \iota)] : \tau}$$

REL-RAND-TAPE-EMPTY-L
$$\frac{\iota \hookrightarrow (N, \varepsilon) \qquad \forall n \leq N.\, \iota \hookrightarrow (N, \varepsilon) \twoheadrightarrow\!\!* \ \Delta \vDash_{\mathcal{E}} E[n] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} E[\mathsf{rand}(N, \iota)] \precsim e_2 : \tau}$$

REL-RAND-TAPE-EMPTY-R
$$\frac{\iota \hookrightarrow_{\mathsf{s}} (N, \varepsilon) \qquad \forall n \leq N.\, \iota \hookrightarrow_{\mathsf{s}} (N, \varepsilon) \twoheadrightarrow\!\!* \ \Delta \vDash_{\mathcal{E}} e_1 \precsim E[n] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[\mathsf{rand}(N, \iota)] : \tau}$$

Reasoning with tapes: asynchronous couplings

$$\frac{f \text{ bijection} \qquad \iota \hookrightarrow (N, \vec{n}) \qquad \forall n \leq N.\, \iota \hookrightarrow (N, \vec{n} \cdot n) \twoheadrightarrow\!\!\!\!* \; \Delta \vDash_{\mathcal{E}} e_1 \precsim E[f(n)] : \tau}{\Delta \vDash_{\mathcal{E}} e_1 \precsim E[\,\mathsf{rand}(N)] : \tau}\;\textsc{rel-couple-tape-l}$$

$$\frac{f \text{ bijection} \qquad \iota \hookrightarrow_{\mathsf{s}} (N, \vec{n}) \qquad \forall n \leq N.\, \iota \hookrightarrow_{\mathsf{s}} (N, \vec{n} \cdot f(n)) \twoheadrightarrow\!\!\!\!* \; \Delta \vDash_{\mathcal{E}} E[n] \precsim e_2 : \tau}{\Delta \vDash_{\mathcal{E}} E[\,\mathsf{rand}(N)] \precsim e_2 : \tau}\;\textsc{rel-couple-tape-r}$$

$$\frac{\begin{array}{c}\iota \hookrightarrow (N, \vec{n}) \qquad \iota' \hookrightarrow_{\mathsf{s}} (N, \vec{n}') \\ f \text{ bijection} \\ \forall n \leq N.\, \iota \hookrightarrow (N, \vec{n} \cdot n) * \iota' \hookrightarrow_{\mathsf{s}} (N, \vec{n}' \cdot f(n)) \twoheadrightarrow\!\!\!\!* \; \Delta \vDash_{\mathcal{E}} e_1 \precsim e_2 : \tau\end{array}}{\Delta \vDash_{\mathcal{E}} e_1 \precsim e_2 : \tau}\;\textsc{rel-couple-tapes}$$

$$\frac{\iota \hookrightarrow_{\mathsf{s}} (N, \varepsilon) \qquad \forall n \leq N.\, \Delta \vDash_{\mathcal{E}} E[n] \precsim E'[n] : \tau}{\Delta \vDash_{\mathcal{E}} E[\,\mathsf{rand}(N)] \precsim E'[\,\mathsf{rand}(N, \iota)] : \tau}\;\textsc{rel-rand-erase-r}$$

$$\frac{\iota \hookrightarrow (N, \varepsilon) \qquad \forall n \leq N.\, \Delta \vDash_{\mathcal{E}} E[n] \precsim E'[n] : \tau}{\Delta \vDash_{\mathcal{E}} E[\,\mathsf{rand}(N, \iota)] \precsim E'[\,\mathsf{rand}(N)] : \tau}\;\textsc{rel-rand-erase-l}$$

Mechanization demo

ElGamal security reduction, II

$PK_{real} \simeq_{\text{ctx}} PK_{real}^{tape}$

PK_{real}	PK_{real}^{tape}
	let β := tape(n) in
let sk := rand(n) in	let sk := rand(n) in
let pk := g^{sk} in	let pk := g^{sk} in
let $count$:= ref 0 in	let $count$:= ref 0 in
let $query = \lambda\ msg$.	let $query = \lambda\ msg$.
if ! $count \neq 0$ then	if ! $count \neq 0$ then
None	None
else	else
$count \leftarrow 1$;	$count \leftarrow 1$;
let b = rand(n) in	let b = rand(n, β) in
let $B = g^b$ in	let $B = g^b$ in
	let $C = pk^b$ in
let $X = msg \cdot pk^b$ in	let $X = msg \cdot C$ in
Some (B, X)	Some (B, X)
in $(pk, query)$	in $(pk, query)$

ElGamal security reduction, II

$PK^{tape}_{real} \qquad \simeq_{ctx} \qquad \mathcal{C}[DH_{real}]$

$$
\begin{array}{l}
\text{let } \beta := \text{tape}(n) \text{ in} \\
\text{let } sk := \text{rand}(n) \text{ in} \\
\text{let } pk := g^{sk} \text{ in} \\
\\
\text{let } count := \text{ref } 0 \text{ in} \\
\text{let } query = \lambda\ msg. \\
\quad \text{if } !\,count \neq 0 \text{ then} \\
\quad\quad \text{None} \\
\quad \text{else} \\
\quad\quad count \leftarrow 1; \\
\quad\quad \text{let } b = \text{rand}(n, \beta) \text{ in} \\
\quad\quad \text{let } B = g^b \text{ in} \\
\quad\quad \text{let } C = pk^b \text{ in} \\
\quad\quad \text{let } X = msg \cdot C \text{ in} \\
\quad\quad \text{Some } (B, X) \\
\text{ in } (pk, query)
\end{array}
$$

$$
\begin{array}{l}
\text{let } (pk, B, C) = \\
\quad \text{let } a := \text{rand}(n) \text{ in} \\
\quad \text{let } b := \text{rand}(n) \text{ in} \\
\quad (g^a, g^b, g^{ab}) \text{ in} \\
\text{let } count := \text{ref } 0 \text{ in} \\
\text{let } query = \lambda\ msg. \\
\quad \text{if } !\,count \neq 0 \text{ then} \\
\quad\quad \text{None} \\
\quad \text{else} \\
\quad\quad count \leftarrow 1; \\
\\
\\
\\
\quad\quad \text{let } X := msg \cdot C \text{ in} \\
\quad\quad \text{Some } (B, X) \\
\text{in } (pk, query)
\end{array}
$$

Current work: reasoning about approximate correctness

- program logic:

$$\boxed{up\_to\ \varepsilon}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

Current work: reasoning about approximate correctness

- program logic:

$$\boxed{up\_to\ \varepsilon}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

- adequacy:

$$\forall \sigma, P \Rightarrow \sum_{\{v \in Val \mid \neg Q(v)\}} \mathrm{exec}(\sigma, e)(v) < \varepsilon$$

Current work: reasoning about approximate correctness

- program logic:

$$\boxed{up\_to\ \varepsilon}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

- adequacy:

$$\forall \sigma, P \Rightarrow \sum_{\{v \in Val \mid \neg Q(v)\}} \mathrm{exec}(\sigma, e)(v) < \varepsilon$$

- examples:
  - cryptographic keys are hard to guess
  - hash collisions are unlikely
  - rejection samplers with arbitrary precision

Current work: reasoning about approximate refinement

- program logic:

$$\boxed{\mathit{up\_to}\ \varepsilon}^{\gamma} \vdash \{P\}\, e_1 < e_2\, \{Q\}$$

Current work: reasoning about approximate refinement

- program logic:

$$\boxed{\mathit{up\_to}\ \varepsilon}^{\gamma} \vdash \{P\}\, e_1 < e_2\, \{Q\}$$

- adequacy:
  If P holds, then for all σ_1, σ_2, there exists an ε-approximate left-partial Q-coupling between $\mathrm{exec}(\sigma_1, e_1)$ and $\mathrm{exec}(\sigma_2, e_2)$.

Current work: reasoning about approximate refinement

- program logic:

$$\boxed{\mathit{up\_to}\ \varepsilon}^{\gamma} \vdash \{P\}\, e_1 < e_2\, \{Q\}$$

- adequacy:
  If P holds, then for all σ_1, σ_2, there exists an ε-approximate left-partial Q-coupling between $\mathrm{exec}(\sigma_1, e_1)$ and $\mathrm{exec}(\sigma_2, e_2)$.

- Examples
  - from crypto: PRP/PRF switching lemma
  - rejection samplers for non-uniform distributions?
  - differential privacy?

Current work: reasoning about termination-preserving refinement

- program logic:

$$\vdash_{\mathcal{M}} \{\mathsf{True}\}\, e\, \{\lambda\_.\, \mathsf{True}\}$$

Current work: reasoning about termination-preserving refinement

- program logic:

$$\vdash_{\mathcal{M}} \{\mathsf{True}\}\ e\ \{\lambda\_.\ \mathsf{True}\}$$

- adequacy:

$$\forall \sigma, \qquad \sum_{a \in A} \mathrm{exec}(\mathcal{M})(a) \ \leq \ \sum_{v \in Val} \mathrm{exec}(\sigma, e)(v)$$

Current work: reasoning about termination-preserving refinement

- program logic:

$$\vdash_{\mathcal{M}} \{\mathsf{True}\}\, e\, \{\lambda\_.\, \mathsf{True}\}$$

- adequacy:

$$\forall \sigma, \qquad \sum_{a \in A} \mathsf{exec}(\mathcal{M})(a) \;\leq\; \sum_{v \in Val} \mathsf{exec}(\sigma, e)(v)$$

- examples
  - lazily sampled, infinite precision real numbers
  - treap data structures
  - iterated Markov chains
  - task schedulers?
  - implementations of stochastic processes?

Future work

- Reasoning about expected running time
  - program logic?

$$\boxed{\texttt{fuel}\ c}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

Future work

- Reasoning about expected running time
  - program logic?

$$\boxed{\texttt{fuel}\ c}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

  - adequacy?

    If P holds, then for all σ, the expected runtime of (σ, e) is c.

Future work

- Reasoning about expected running time
  - program logic?

$$\boxed{\texttt{fuel}\ c}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

  - adequacy?

    If P holds, then for all σ, the expected runtime of (σ, e) is c.

  - Examples
    - algorithms: quicksort?
    - data structures: treaps? skip lists?
    - rejection samplers?

Future work

- Reasoning about expected running time
  - program logic?

$$\boxed{\texttt{fuel}\ c}^{\gamma} \vdash \{P\}\, e\, \{Q\}$$

  - adequacy?

    If P holds, then for all σ, the expected runtime of (σ, e) is c.

  - Examples
    - algorithms: quicksort?
    - data structures: treaps? skip lists?
    - rejection samplers?

- Reasoning about distributed randomized system
  - dining philosophers, consensus, multi-party computation?

Joint work with

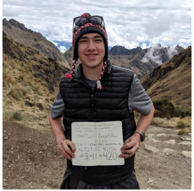
Alejandro Aguirre

Simon Gregersen

Joseph Tassarotti

Lars Birkedal

Hei Li



Markus de Medeiros