
CSLR: A Calculus for Cryptographic Proofs

Yu Zhang

Institute of Software, Chinese Academy of Sciences



Joint work with *David Nowak* (AIST, Tokyo)

April 15, 2010

Barbizon, France

Computational indistinguishability

- ▶ A notion of **observational equivalence**: crypto-systems are programs, and equivalence says the closeness between probabilistic distributions.
- ▶ Well studied in PL and logic, supported by many proof techniques.



- ▶ A suitable formal language with notions of probabilistic complexity.

Outline

The computational SLR

CSLR — syntax and semantics

Complexity results

Cryptographic constructions in CSLR

The CSLR proof system

Security notions in CSLR

The proof system

Cryptographic examples

Game-based proofs in CSLR

Game indistinguishability

Conclusion

The computational SLR

Hofmann's SLR system

- ▶ A functional language characterizing PTIME computations through typing.
- ▶ An implementation of Bellantoni and Cook's **safe recursion**:
 - It's recursion on (binary) notation.
 - Variables are divided into **normal** and **safe** variables: $f(\vec{x}; \vec{y})$.
 - **Recursive calls via safe variables**:

$$f(0, \vec{y}; \vec{z}) = g(\vec{y}; \vec{z})$$

$$f(x, \vec{y}; \vec{z}) = h(x, \vec{y}; f(\lfloor \frac{x}{2} \rfloor, \vec{y}; \vec{z}), \vec{z})$$

- A recursion characterization of PTIME **without any explicit bound**.

Hofmann's SLR system

- ▶ An alternative syntax using **modality**.

- $\Box(\tau)$ are types for normal variables.

$$\frac{\Gamma \vdash e : \Box(\tau)}{\Gamma \vdash e : \tau} \quad \frac{\Gamma \vdash e : \tau \quad \Gamma(x) = \Box(_) \text{ for all } x \in FV(e)}{\Gamma \vdash e : \Box(\tau)}$$

- Safe recursor:

$$\text{rec} : \mathbb{N} \rightarrow (\Box(\mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \Box(\mathbb{N}) \rightarrow \mathbb{N}$$

- ▶ Higher-order recursive calls must be used **linearly**.

$$\text{rec}_\tau : \tau \multimap (\Box(\mathbb{N}) \rightarrow \tau \multimap \tau) \rightarrow \Box(\mathbb{N}) \rightarrow \tau$$

- Higher-order recursion is not necessary for characterizing PTIME computations, but it eases programming.

CSLR — types

An extension of the non-polymorphic SLR with **monadic types**:

τ, τ', \dots	$::=$	Bits	bitstrings
		$\tau \multimap \tau'$	linear functions
		$\tau \rightarrow \tau'$	nonlinear, nonmodal functions
		$\Box \tau \rightarrow \tau'$	modal (normal) functions
		$\mathsf{T}\tau$	probabilistic computations
		\dots	

- \Box itself is NOT a type constructor in SLR.
- Constructor T is from Moggi's computational λ -calculus.
- Subtyping:

$$\tau \multimap \tau' \leq \tau \rightarrow \tau' \leq \Box \tau \rightarrow \tau' \quad \text{Bits} \rightarrow \tau \leq \text{Bits} \multimap \tau$$

Bistrings can be duplicated without breaking linearity.

CSLR — expressions

Expressions:

e, e', \dots	$::=$	nil	empty bitstring
		$B_0 \mid B_1$	bit successor
		rec_τ	safe recursor
		rand	oracle bit
		$\text{val}(e)$	trivial (deterministic) computations
		$\text{bind } x = e \text{ in } e'$	sequential computations
		\dots	

- We operate directly on [bitstrings](#), instead of numbers.
- Probabilistic computations are formulated in Moggi's framework.

CSLR — type system

Typing contexts assign **aspects** as well as types to variables:

$$x_1 :^{a_1} \tau_1, \dots, x_n :^{a_n} \tau_n$$

- Aspects specify the way how variables can be used in programs.
- Three (ordered) aspects corresponding to the function spaces:
(non-linear, modal) \leq (non-linear, non-modal) \leq (linear, non-modal)
- Higher aspects mean more restriction on variables.

CSLR — typing rules

$$\frac{}{\Gamma \vdash \mathbf{rec}_\tau : \tau \multimap \Box(\Box\mathbf{N} \rightarrow \tau \multimap \tau) \rightarrow \Box\mathbf{N} \rightarrow \tau} \quad \frac{\Gamma, x :^a \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \xrightarrow{a} \tau'}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : \tau \xrightarrow{a} \tau' \quad \Gamma, \Delta_2 \vdash e_2 : \tau \quad \Gamma \text{ nonlinear} \quad a' \leq a \text{ for all } x :^{a'} \tau'' \in \Gamma, \Delta_2}{\Gamma, \Delta_1, \Delta_2 \vdash e_1 e_2 : \tau'}$$

... ..

$$\frac{}{\Gamma \vdash \mathbf{rand} : \mathbf{TBits}} \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{val}(e) : \mathbf{T}\tau}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : \mathbf{T}\tau \quad \Gamma, \Delta_2, x :^a \tau \vdash e_2 : \mathbf{T}\tau' \quad \Gamma \text{ nonlinear} \quad a' \leq a \text{ for all } x :^{a'} \tau'' \in \Gamma, \Delta_1}{\Gamma, \Delta_1, \Delta_2 \vdash \mathbf{bind } x = e_1 \text{ in } e_2 : \mathbf{T}\tau'}$$

CSLR — semantics

- ▶ The **set-theoretic** semantics:
 - The set \mathbb{B} of all bitstrings (including the empty one) for interpreting Bits.
 - We do not distinguish between the three sorts of function spaces.
 - $\llbracket \text{rec} \rrbracket$ defines the safe recursion scheme.
- ▶ A probabilistic monad for interpreting probabilistic computations

$$\begin{aligned}\llbracket \top \tau \rrbracket &= \llbracket \tau \rrbracket \rightarrow [0, 1] \\ \llbracket \text{rand} \rrbracket &= \{(0, \frac{1}{2}), (1, \frac{1}{2})\} \\ \llbracket \text{val}(e) \rrbracket \rho &= \{(\llbracket e \rrbracket \rho, 1)\} \\ \llbracket \text{bind } x = e_1 \text{ in } e_2 \rrbracket \rho &= \underline{\lambda} v. \sum_{v' \in \llbracket \tau \rrbracket} \llbracket e_2 \rrbracket \rho[x \mapsto v'](v) \times \llbracket e_1 \rrbracket \rho(v')\end{aligned}$$

- The monad defined using measures [Ramsey & Pfeffer '02], but simplified here by using **mass functions**.

Complexity results on (computational) SLR

- ▶ Hofmann's theorem:
 - The set-theoretic interpretations of well typed SLR terms of type $\Box \text{Bits} \rightarrow \text{Bits}$ are exactly **P**TIME functions.
- ▶ Theorem of Mitchell et al. (adapted):
 - The set-theoretic interpretations of well typed CSLR terms of type $\Box \text{Bits} \rightarrow \text{TBits}$ are exactly **PPT** functions.
 - The language of Mitchell et al. does not have computation types, but their proof applies to CSLR.

CSLR functions for bitstrings

Many bitstring operations can be defined in CSLR:

- The random bitstring generation

$rs \stackrel{\text{def}}{=} \lambda x : \text{Bits} . \text{rec}(\text{val}(\text{nil}), h_{rs}, x)$, where

$$h_{rs} \stackrel{\text{def}}{=} \lambda m . \lambda r . \text{bind } b = \text{rand in bind } u = r \text{ in} \\ \text{case}(b, \langle \text{val}(\text{nil}), \lambda x . \text{val}(\text{B}_0 u), \lambda x . \text{val}(\text{B}_1 u) \rangle)$$

- The string concatenation $conc \stackrel{\text{def}}{=} \lambda x . \lambda y . \text{rec}(y, h_{conc}, x)$, where

$$h_{conc} \stackrel{\text{def}}{=} \lambda m . \lambda r . \text{case}(m, \langle r, \lambda x . \text{B}_0 r, \lambda x . \text{B}_1 r \rangle).$$

- Head function $hd \stackrel{\text{def}}{=} \lambda x . \text{case}(x, \langle \text{nil}, \lambda y . 0, \lambda y . 1 \rangle).$

Tail function $tl \stackrel{\text{def}}{=} \lambda x . \text{case}(x, \langle \text{nil}, \lambda y . y, \lambda y . y \rangle).$

-

All are well-typed CSLR terms.

Cryptographic constructions in CSLR

- ▶ Goldreich and Micali's pseudorandom construction:

$$G \stackrel{\text{def}}{=} \lambda l . \lambda s . \text{rec}(\text{nil}, \lambda m . \lambda r . r \bullet \text{head}(g_1(R'(s, m))), l)$$

where $R' \stackrel{\text{def}}{=} \lambda l . \lambda s . \text{rec}(s, \lambda m . \lambda r . \text{tail}(g_1(\text{pref}(r, s))), l)$.

G is of type $\square \text{Bits} \rightarrow \square \text{Bits} \rightarrow \text{Bits}$.

- ▶ Blum-Blum-Shub pseudorandom construction:

$$BBS \stackrel{\text{def}}{=} \lambda l . \lambda s . \text{bbsrec}(l, s^2)$$

where bbsrec is defined as

$$\text{bbsrec} \stackrel{\text{def}}{=} \lambda l . \text{rec}(\lambda x . \text{nil}, \lambda m . \lambda r . \lambda x . \text{parity}(x) \bullet r(x^2), l).$$

BBS is of type $\square \text{Bits} \rightarrow \square \text{Bits} \rightarrow \text{Bits}$.

Cryptographic constructions in CSLR

► El-Gamal encryption scheme:

- The key generation:

$$KG \stackrel{\text{def}}{=} \lambda \eta . \text{bind } x = \mathbf{zrand}(q) \text{ in val}(\gamma^x, x)$$

KG is of type $\square \text{Bits} \rightarrow T(\text{Bits} \times \text{Bits})$.

- The encryption:

$$Enc \stackrel{\text{def}}{=} \lambda \eta . \lambda pk . \lambda m . \text{bind } y = \mathbf{zrand}(q) \text{ in val}(\gamma^y, pk^y * m))$$

Enc is of type $\square \text{Bits} \rightarrow \text{Bits} \rightarrow \text{Bits} \rightarrow T(\text{Bits} \times \text{Bits})$.

- The decryption:

$$Dec \stackrel{\text{def}}{=} \lambda \eta . \lambda sk . \lambda c . \text{proj}_2(c) * (\text{proj}_1(c)^{sk})^{-1}$$

Dec is of type $\square \text{Bits} \rightarrow \text{Bits} \rightarrow \text{Bits} \rightarrow \text{Bits}$.

The CSLR proof system

Computational indistinguishability (in CSLR)

Two CSLR programs f_1, f_2 of type $\square\text{Bits} \rightarrow \tau$ are **computationally indistinguishable** (written as $f_1 \simeq f_2$) if

- for every well typed CSLR program \mathcal{A} (adversary) of type $\square\text{Bits} \rightarrow \tau \rightarrow \text{TBits}$,
- for every positive polynomial p (SLR term of type $\square\text{Bits} \rightarrow \text{Bits}$),
- for every **sufficiently long** bitstring η ,

$$|\mathbf{Pr}[\llbracket \mathcal{A}(\eta, f_1(\eta)) \rrbracket = 1] - \mathbf{Pr}[\llbracket \mathcal{A}(\eta, f_2(\eta)) \rrbracket = 1]| < \frac{1}{|p(\eta)|}$$

- The adversary is *feasible* iff it is well typed.

Security notions by computational indistinguishability

- ▶ **Pseudorandomness**: a deterministic function G (of type $\square\text{Bits} \rightarrow \text{Bits}$) is a PRG if $|G(s)| > |s|$ for every s and

$$\lambda x.\text{bind } s = \mathbf{rs}(x) \text{ in val}(G(s)) \simeq \lambda x.\mathbf{rs}(G(x))$$

- ▶ **Next-bit unpredictability**: a deterministic function F (of type $\square\text{Bits} \rightarrow \text{Bits}$) is *next-bit unpredictable* if $|F(s)| > |s|$ for every s and

$$\begin{aligned} & \lambda \eta.\text{bind } s = \mathbf{rs}(\eta) \text{ in val}(F(s)) \\ \simeq & \lambda \eta.\text{bind } s = \mathbf{rs}(\eta) \text{ in bind } b = \text{rand} \text{ in val}(b \bullet \mathbf{tail}(F(s))) \end{aligned}$$

- ▶ **(Strong) semantic security**:

$$\begin{aligned} & \lambda \eta.\text{bind } k = \mathbf{KG}(\eta) \text{ in val}(\lambda m_0.\lambda m_1.\mathbf{Enc}(\eta, k, m_0)) \\ \simeq & \lambda \eta.\text{bind } k = \mathbf{KG}(\eta) \text{ in val}(\lambda m_0.\lambda m_1.\mathbf{Enc}(\eta, k, m_1)) \end{aligned}$$

The proof system — internal rules

Rules justifying **program equivalence**.

- Standard axioms and rules in λ -calculus:

$$\frac{}{e \equiv e} \quad \frac{}{(\lambda x.e)e' \equiv e[e'/x]} \quad \frac{e \equiv e'}{\lambda x.e \equiv \lambda x.e'} \quad \dots\dots$$

- Axioms and rules for probabilistic computations:

$$\frac{}{\text{bind } x = \text{val}(e_1) \text{ in } e_2 \equiv e_2[e_1/x]}$$

$$\frac{}{\text{bind } x = (\text{bind } y = e_1 \text{ in } e_2) \text{ in } e_3 \equiv \text{bind } y = e_1 \text{ in } \text{bind } x = e_2 \text{ in } e_3}$$

$$\frac{e_1 \equiv e'_1 \quad e_2 \equiv e'_2}{\text{bind } x = e_1 \text{ in } e_2 \equiv \text{bind } x = e'_1 \text{ in } e'_2}$$

.....

Two probabilistic programs are equivalent if they produce the same distribution.

The proof system — internal rules

Rules justifying **computational indistinguishability**.

$$\frac{\vdash e_1 : \square\text{Bits} \rightarrow \tau \quad \vdash e_2 : \square\text{Bits} \rightarrow \tau \quad e_1 \equiv e_2}{e_1 \simeq e_2} \text{EQUIV}$$

$$\frac{\vdash e_i : \square\text{Bits} \rightarrow \tau (i = 1, 2, 3) \quad e_1 \simeq e_2 \quad e_2 \simeq e_3}{e_1 \simeq e_3} \text{TRANS-INDIST}$$

$$\frac{x :^m \text{Bits}, y :^m \tau \vdash e : \tau \quad \vdash e_i : \square\text{Bits} \rightarrow \tau' (i = 1, 2) \quad e_1 \simeq e_2}{\lambda x. \text{bind } y = e_1(x) \text{ in } e \simeq \lambda x. \text{bind } y = e_2(x) \text{ in } e} \text{SUB}$$

$$\frac{x :^m \text{Bits}, n :^m \text{Bits} \vdash e : \tau \quad \lambda n. e[u/x] \text{ is numerical for all } u \\ \lambda x. e[i(x)/n] \simeq \lambda x. e[\text{B}_1 i(x)/n] \text{ for all canonical } i \text{ such that } |i| < |p|}{\lambda x. e[\text{nil}/n] \simeq \lambda x. e[p(x)/n]} \text{H-IND}$$

The proof system — lemmas (as external rules)

► An extendable set of lemmas:

- *RS-EQUIV*: If $|u| = |u'|$, then $rs(u) \equiv rs(u')$.
- *RS-CONCAT*:

$$\text{bind } x = rs(u) \text{ in bind } y = rs(u') \text{ in val}(x \bullet y) \equiv rs(u \bullet u')$$

- *RS-CUT*:

$$\text{bind } x = rs(u) \text{ in val}(x - u') \equiv rs(u - u')$$

- *RS-COMMUT*:

$$\begin{aligned} & \text{bind } x = rs(u) \text{ in bind } y = rs(v) \text{ in val}(x \bullet y) \\ \equiv & \text{bind } x = rs(u) \text{ in bind } y = rs(v) \text{ in val}(y \bullet x) \end{aligned}$$

–

- These lemmas can be proved using the previous two sets of rules, but they are seen and used as **external rules** of the proof system.

Soundness

- ▶ Soundness theorem about program equivalence rules:
 - If $e_1 \equiv e_2$ is provable, then $\llbracket e_1 \rrbracket \rho = \llbracket e_2 \rrbracket \rho$.
 - The probability monad justifies the soundness of axioms of probabilistic computations.
- ▶ Soundness theorem about computational indistinguishability:
 - If $e_1 \simeq e_2$ is provable, then e_1 and e_2 are computationally indistinguishable.

Example: Goldreich and Micali's PRG

- ▶ Theorem: For every polynomial p , $\lambda x.G(x, p(x))$ is a PRG.
 - We prove $\lambda x.\text{bind } s = \mathbf{rs}(x) \text{ in val}(G(s, p(s))) \simeq \lambda x.\mathbf{rs}(p(x))$.
 - The proof follows the traditional hybrid technique, with the hybrid function $H \stackrel{\text{def}}{=} \lambda x . \lambda y . \lambda n . (y - n) \bullet G(x, n)$.

$$\begin{aligned} & \lambda x . \text{bind} (u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x))) \text{ in val}(H(u_1, u_2, \text{Bi}(x))) \\ \equiv & \lambda x . \text{bind} (u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x))) \text{ in val}((u_2 - \text{Bi}(x)) \bullet G(u_1, \text{Bi}(x))) \\ \simeq & \lambda x . \text{bind} (b = \text{rand}, u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x))) \text{ in val}((u_2 - \text{Bi}(x)) \bullet b \bullet G(u_1, i(x))) \\ & \quad \text{(by Lemma 14 and the rule SUB)} \\ \equiv & \lambda x . \text{bind} (b = \text{rand}, u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x) - \text{Bi}(x))) \text{ in val}(u_2 \bullet b \bullet G(u_1, i(x))) \\ & \quad \text{(by the rule RS-CUT, as } |\text{Bi}(x)| = |i(x)| + 1 \leq |p(x)| = |u_2|) \\ \equiv & \lambda x . \text{bind} (u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}((p(x) - \text{Bi}(x)) \bullet 1)) \text{ in val}(u_2 \bullet G(u_1, i(x))) \\ & \quad \text{(by the rule RS-CONCAT)} \\ \equiv & \lambda x . \text{bind} (u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x) - i(x))) \text{ in val}(u_2 \bullet G(u_1, i(x))) \\ & \quad \text{(because } |(p(x) - \text{Bi}(x)) \bullet 1| = |p(x) - i(x)| - 1 + 1 = |p(x) - i(x)|) \\ \equiv & \lambda x . \text{bind} (u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x))) \text{ in val}((u_2 - i(x)) \bullet G(u_1, i(x))) \\ & \quad \text{(by the rule RS-CUT)} \\ \equiv & \lambda x . \text{bind} (u_1 = \mathbf{rs}(x), u_2 = \mathbf{rs}(p(x))) \text{ in val}(H(u_1, u_2, i(x))) \end{aligned}$$

Game-based proofs in CSLR

Game indistinguishability

- ▶ Many cryptographers advocate **game-based proofs**:
 - Security criteria are specified using **games**.
 - Crypto proofs are constructed as sequences of close games.
- ▶ In CSLR, a **game** is a function of type $\square\text{Bits} \rightarrow (\square\text{Bits} \rightarrow \tau) \rightarrow \text{TBits}$ that returns one bit.
- ▶ Two CSLR games g_1, g_2 are **game indistinguishable** (written as $g_1 \cong g_2$) if
 - for every well typed CSLR program \mathcal{A} (adversary) of type $\square\text{Bits} \rightarrow \tau$,
 - for every positive polynomial p (SLR term of type $\square\text{Bits} \rightarrow \text{Bits}$),
 - for every **sufficiently long** bitstring η ,

$$|\mathbf{Pr}[\llbracket g_1(\eta, \mathcal{A}) \rrbracket = 1] - \mathbf{Pr}[\llbracket g_2(\eta, \mathcal{A}) \rrbracket = 1]| < \frac{1}{|p(\eta)|}$$

Security notions by game indistinguishability

► Next-bit unpredictability

$$\begin{aligned} \lambda\eta. \lambda\mathcal{A}. \quad & \text{bind } s = \mathbf{rs}(\eta) \text{ in} \\ & \text{let } u = F(s) \text{ in} \\ & \text{bind } b = \mathcal{A}(\eta, \mathbf{tail}(u)) \text{ in} \\ & \text{val}(b \stackrel{?}{=} \mathbf{head}(u)) \end{aligned} \cong \lambda\eta. \lambda\mathcal{A}. \text{rand}$$

► Semantic security

$$\begin{aligned} \lambda\eta. \lambda\mathcal{A}. \quad & \text{bind } (pk, sk) = \mathbf{KG}(\eta) \text{ in} \\ & \text{bind } (m_0, m_1, \mathcal{A}') = \mathcal{A}(\eta, pk) \text{ in} \\ & \text{bind } b = \text{rand} \text{ in} \\ & \text{bind } c = \mathbf{Enc}(\eta, pk, m_b) \text{ in} \\ & \text{bind } b' = \mathcal{A}'(c) \text{ in} \\ & \text{val}(b' \stackrel{?}{=} b) \end{aligned} \cong \lambda\eta. \lambda\mathcal{A}. \text{rand}$$

Proving game indistinguishability

- ▶ Theorem: Computational indistinguishability implies game indistinguishability.
 - The proof system of CSLR applies to game-based proofs.
- ▶ We have formalized in CSLR:
 - Game-based proof of semantic security of El-Gamal encryption.
 - Based on the [binary implementation](#) — asymptotic uniform sampling.
- ▶ CSLR₊:
 - Extends CSLR with general uniform sampling and (super-polynomial time) constants.
 - Adversaries are always defined in CSLR.
 - We formalized game-based proofs of next-bit unpredictability of Blum-Blum-Shub PSG.

Conclusion

Related work

- ▶ **PPC by Mitchell et al.:** CCS-like language with bound replications; asymptotic bisimulation for computational indistinguishability.
- ▶ **Logics by Impagliazzo and Kapron:** non-standard arithmetic model; explicit reasoning about probability.
- ▶ Automated verification of **game-based proofs**:
 - Nowak: shallow embedding (crypto schemes as distributions); implemented in Coq.
 - Barthe et al.: deep embedding with an imperative language; relational Hoare logic; implemented in Coq.
 - Backes et al.: functional language with references and events; implemented in Isabelle/HOL.
- ▶ **Blanchet's CryptoVerif:** CCS-like language; automated generation of game sequences.

Conclusion

- ▶ Cryptography in a perspective of high-order languages.
 - **Implicit complexity** helps to manage complexity issues.
- ▶ Future work:
 - More applications in cryptography.
 - Computational verification of protocols: higher order functions are already there.
 - Automated proof checking.
 - Theoretical issues ...