The Computational SLR: A Calculus for Verifying Cryptographic Proofs

Yu Zhang

Institute of Software
Chinese Academy of Sciences

BASICS’09, Shanghai, China
October 13, 2009
Background

- Formal verification of security protocols
  - from the symbolic model to the computational model.

- In cryptography, bugs are continuously found in crypto proofs, which sometimes take time.
  - OAEP scheme was initially believed to be IND-CCA2 secure, but was proved not, after 7 years.

“Many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor.”

— Bellare & Rogaway 2004
Proof. Let us define $K = 2^k$, $H_s = \left\lfloor \frac{2^{s+1}-1}{K} \right\rfloor$ for $s \in [0, K - 1]$. Let denote by $e_p$ the following character of $\mathbb{Z}_p$: for all $y \in \mathbb{Z}_p$, $e_p(y) = e^{\frac{yq}{p}} \in \mathbb{C}^*$. The character $e_p$ is an homomorphism from $(\mathbb{Z}_p, +)$ in $(\mathbb{C}^*, \cdot)$. Since

$$\frac{1}{p} \sum_{a=0}^{p-1} e_p(a(g^{x} - s - Ku)) = \mathbb{1}(a, s, u),$$

where $\mathbb{1}(a, s, u)$ is the characteristic function which is equal to 1 if $g^x = s + Ku \mod p$ and 0 otherwise, we have:

$$\Pr_{X \in \mathbb{G}}[c_b(X) = a] = \frac{1}{q} \times \left| \{(a, s, u) \in [0, q-1] \times [0, H_s] \mid g^x = s + Ku \mod p \} \right|$$

$$= \frac{1}{qp} \sum_{a=0}^{q-1} \sum_{u=0}^{H_s} \sum_{a=0}^{p-1} e_p(a(g^{x} - s - Ku)).$$

Let us change the order of the sums, and split sum on the $a$’s in two terms:

1. the first one comes from the case $a = 0$, and is equal to $(H_s + 1)/p$, that is approximately $1/2^k$,
2. the second one comes from the rest, and will be the principal term in the statistical distance in which we can separate sums over $x$ and $u$.

Twice the statistical distance, that is $2\Delta$, is equal to:

$$\sum_{x \in \{0,1\}^k} \left| \Pr_{X \in \mathbb{G}}[c_b(X) = a] - 1/2^k \right|$$

$$\leq \sum_{x \in \{0,1\}^k} \left| \frac{H_s + 1}{p} - \frac{1}{2^k} \right| + \sum_{x \in \{0,1\}^k} \frac{1}{q^2} \sum_{a=1}^{p-1} \left| \sum_{u=0}^{H_s} e_p((ag^x) - aKu) \right|.$$

For the first term, we notice that $|(H_s + 1)/p - 1/2^k| \leq 1/p$, since $K = 2^k$, $H_s = \left\lfloor \frac{2^{s+1}-1}{K} \right\rfloor$ and:

$$-\frac{1}{p} \leq -\frac{1+s}{Kp} \leq \left(1 + \left\lfloor \frac{p-1-s}{K} \right\rfloor \right)^{-1} \leq \frac{K - (1+s)}{Kp} \leq \frac{1}{p}.$$
Many security criteria in cryptography are defined using computational indistinguishability.

- It is a notion of observational equivalence: crypto-systems are programs.
- Well studied in PL and logic, supported by many proof techniques.

Checking the damned proof is just killing me!

But, isn't that observational equivalence between programs?
Outline

➢ The computational SLR
➢ The proof system for computational indistinguishability
➢ Game-based proofs in CSLR
➢ Conclusion
Hofmann’s SLR system

✦ A functional language characterizing PTIME computations through typing.
✦ An implementation of Bellantoni and Cook’s safe recursion:
  – Variables are divided into normal and safe variables: \( f(\vec{x}; \vec{y}) \).
  – Recursive calls via safe variables:
    
    \[
    f(0, \vec{y}; \vec{z}) = g(\vec{y}; \vec{z}) \\
    f(x, \vec{y}; \vec{z}) = h(x, \vec{y}; f(\lceil \frac{x}{2} \rceil, \vec{y}; \vec{z}), \vec{z})
    \]

✦ \( \Box(\tau) \) are types for normal variables.

\[
\begin{align*}
\Gamma \vdash e : \Box(\tau) & \quad \Gamma \vdash e : \tau \\
\Gamma \vdash e : \tau & \quad \Gamma(x) = \Box(_) \text{ for all } x \in FV(e) \\
\end{align*}
\]

  – Safe recursor: \( \text{rec} : \mathbb{N} \to (\Box(\mathbb{N}) \to \mathbb{N} \to \mathbb{N}) \to \Box(\mathbb{N}) \to \mathbb{N} \)

✦ Higher-order recursive calls must be used linearly.

\[
\text{rec}_\tau : \tau \to (\Box(\mathbb{N}) \to \tau \to \tau) \to \Box(\mathbb{N}) \to \tau
\]
The computational SLR (CSDLR) — types

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

\[ \tau, \tau', \ldots ::= \text{Bits bitstrings} \]

\[ | \quad \tau \rightarrow \tau' \quad \text{linear functions} \]

\[ | \quad \tau \rightarrow \tau' \quad \text{nonlinear, nonmodal functions} \]

\[ | \quad \Box \tau \rightarrow \tau' \quad \text{modal (normal) functions} \]

\[ | \quad T\tau \quad \text{probabilistic computations} \]

\[ | \quad \ldots \]

- \( \Box \) itself is NOT a type constructor in SLR.
- Constructor \( T \) is from Moggi’s computational \( \lambda \)-calculus.
Expressions:

\[ e, e', \ldots ::= \begin{aligned}
    &\text{nil} &\text{empty bitstring} \\
    | &B_0 \mid B_1 &\text{bit successor} \\
    | &\text{rec}_\tau &\text{safe recursor} \\
    | &\text{rand} &\text{oracle bit} \\
    | &\text{val}(e) &\text{trivial (deterministic) computations} \\
    | &\text{bind } x = e \text{ in } e' &\text{sequential computations} \\
    | \ldots &
\end{aligned} \]

- We operate directly on bitstrings, instead of numbers.
- Probabilistic computations are formulated in Moggi’s framework.
Typing contexts assign aspects as well as types to variables:

\[
x_1 : ^{a_1} \tau_1, \ldots, x_n : ^{a_n} \tau_n
\]

Aspects specify the way how variables can be used in terms.

Typing rules:

\[
\Gamma \vdash \text{rec}_{\tau} : \tau \rightarrow \Box(\Box N \rightarrow \tau \rightarrow \tau) \rightarrow \Box N \rightarrow \tau
\]

\[
\Gamma, x : ^{a} \tau \vdash e : \tau'
\]

\[
\Gamma \vdash \lambda x.e : \tau \xrightarrow{a} \tau'
\]

\[
\Gamma, \Delta_1 \vdash e_1 : \tau \xrightarrow{a} \tau'
\quad \Gamma, \Delta_2 \vdash e_2 : \tau
\]

\[
\Gamma \text{ nonlinear } a' \leq a \text{ for all } x : ^{a'} \tau'' \in \Gamma, \Delta_2
\]

\[
\Gamma, \Delta_1, \Delta_2 \vdash e_1 e_2 : \tau'
\]

\[
\Gamma \vdash e : \tau
\]

\[
\Gamma \vdash \text{val}(e) : \mathcal{T}\tau
\]

\[
\Gamma, \Delta_1 \vdash e_1 : \mathcal{T}\tau
\quad \Gamma, \Delta_2, x : ^{a} \tau \vdash e_2 : \mathcal{T}\tau'
\]

\[
\Gamma \text{ nonlinear } a' \leq a \text{ for all } x : ^{a'} \tau'' \in \Gamma, \Delta_1
\]

\[
\Gamma, \Delta_1, \Delta_2 \vdash \text{bind } x = e_1 \text{ in } e_2 : \mathcal{T}\tau'
\]
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.
- $[\text{rec}]$ defines the safe recursion scheme.

A probabilistic monad for interpreting probabilistic computations

$$
[T \tau] = [\tau] \rightarrow [0, 1] \\
[\text{rand}] = \{(0, \frac{1}{2}), (1, \frac{1}{2})\} \\
[\text{val}(e)] \rho = \{([e] \rho, 1)\} \\
[\text{bind } x = e_1 \text{ in } e_2] \rho = \lambda v. \sum_{v' \in [\tau]} [e_2] \rho[x \mapsto v'](v) \times [e_1] \rho(v')
$$

- The monad defined using measures [Ramsey & Pfeffer '02], but simplified here by using mass functions.
Results on (computational) SLR

- Hofmann’s theorem:
  - The set-theoretic interpretations of well typed SLR terms of type □Bits \(\rightarrow\) Bits are exactly PTIME functions.

- Theorem of Mitchell et al. (adapted):
  - The set-theoretic interpretations of well typed CSLR terms of type □Bits \(\rightarrow\) TBits are exactly PPT functions.
  - The language of Mitchell et al. does not have computation types, but their proof applies to CSLR.
Cryptographic constructions in CSLR

✦ Goldreich and Micali’s pseudorandom construction:

\[ G \overset{\text{def}}{=} \lambda u . \lambda n . \text{rec}(\text{nil}, \lambda m . \lambda r . r \cdot \text{head}(g_1(R'(u, m))), n) \]

where \( R' \overset{\text{def}}{=} \lambda u . \lambda n . \text{rec}(u, \lambda m . \lambda r . \text{tail}(g_1(\text{pref}(r, u))), n). \)

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

✦ Blum-Blum-Shub pseudorandom construction:

\[ BBS \overset{\text{def}}{=} \lambda l . \lambda s . \text{bbsrec}(l, s^2) \]

where \( \text{bbsrec} \) is defined as

\[ \text{bbsrec} \overset{\text{def}}{=} \lambda l . \text{rec}(\lambda x . \text{nil}, \lambda m . \lambda r . \lambda x . \text{parity}(x) \cdot r(x^2), l). \]

\( BBS \) is of type \( \square \text{Bits} \rightarrow \square \text{Bits} \rightarrow \text{Bits} \).
El-Gamal encryption scheme:

- The key generation:
  
  $$KG \overset{\text{def}}{=} \lambda \eta . \text{bind } x = \text{zrand}(q) \text{ in } \text{val}(\gamma^x, x)$$

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

- The encryption:
  
  $$Enc \overset{\text{def}}{=} \lambda \eta . \lambda \text{pk} . \lambda m . \text{bind } y = \text{zrand}(q) \text{ in } \text{val}(\gamma^y, \text{pk}^y \ast m)$$

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

- The decryption:
  
  $$Dec \overset{\text{def}}{=} \lambda \eta . \lambda \text{sk} . \lambda c . \text{proj}_2(c) \ast (\text{proj}_1(c)^{sk})^{-1}$$

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

➢ The computational SLR
➢ The proof system for computational indistinguishability
➢ Game-based proofs in CSLR
➢ Conclusion
Two CSLR programs $f_1, f_2$ of type $\Box \text{Bits} \to \tau$ are computationally indistinguishable (written as $f_1 \simeq f_2$) if

- for every well typed CSLR program $A$ (adversary) of type $\Box \text{Bits} \to \tau \to \text{TBits}$,
- for every positive polynomial $p$ (SLR term of type $\Box \text{Bits} \to \text{Bits}$),
- for every sufficiently long bitstring $\eta$,$$
|\Pr[[A(\eta, f_1(\eta))] = \epsilon] - \Pr[[A(\eta, f_2(\eta))] = \epsilon]| < \frac{1}{|p(\eta)|}
$$

- The adversary is feasible iff it is well typed.
Security notions by computational indistinguishability

- **Pseudorandomness**: a deterministic function $G$ (of type □Bits → Bits) is a PRG if $|G(s)| > |s|$ for every $s$ and

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

- **Next-bit unpredictability**: a deterministic function $F$ (of type □Bits → Bits) is next-bit unpredictable if $|F(s)| > |s|$ for every $s$ and

$$\lambda \eta.\text{bind } s = rs(\eta) \text{ in } \text{val}(F(s))$$

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

- **Semantic security**:

$$\lambda \eta.\text{bind } k = KG(\eta) \text{ in } \text{val}(KG, Enc, Dec, \lambda m_0.\lambda m_1.\text{Enc}(\eta, k, m_0))$$

$$\simeq \lambda \eta.\text{bind } k = KG(\eta) \text{ in } \text{val}(KG, Enc, Dec, \lambda m_0.\lambda m_1.\text{Enc}(\eta, k, m_1))$$
The proof system — internal rules

Rules justifying program equivalence.

• Standard axioms and rules in λ-calculus:

\[
\begin{align*}
  e & \equiv e \\
  (\lambda x.e)e' & \equiv e[e'/x] \\
  \lambda x.e & \equiv \lambda x.e'
\end{align*}
\]

• Axioms and rules for probabilistic computations:

\[
\begin{align*}
  \text{bind } x = \text{val}(e_1) \text{ in } e_2 & \equiv e_2[e_1/x] \\
  \text{bind } x = (\text{bind } y = e_1 \text{ in } e_2) \text{ in } e_3 & \equiv \text{bind } y = e_1 \text{ in } \text{bind } x = e_2 \text{ in } e_3 \\
  e_1 & \equiv e'_1 \\
  e_2 & \equiv e'_2
\end{align*}
\]

\[
\begin{align*}
  \text{bind } x = e_1 \text{ in } e_2 & \equiv \text{bind } x = e'_1 \text{ in } e'_2 \\
  \ldots
\end{align*}
\]

Two probabilistic programs are equivalent if they produce the same distribution.
The proof system — internal rules

Rules justifying computational indistinguishability.

\[ \vdash e_1 : \Box \text{Bits} \rightarrow \text{TBits} \quad \vdash e_2 : \Box \text{Bits} \rightarrow \text{TBits} \quad e_1 \equiv e_2 \]

\[ e_1 \simeq e_2 \]

\[ e_1 \simeq e_2 \quad e_2 \simeq e_3 \]

\[ e_1 \simeq e_3 \]

\[ x : ^n \text{Bits}, y : ^n \text{Bits} \vdash e : \text{TBits} \quad e_1 \simeq e_2 \]

\[ \lambda x . \text{bind } y = e_1(x) \text{ in } e \simeq \lambda x . \text{bind } y = e_2(x) \text{ in } e \]

\[ x : ^n \text{Bits}, n : ^n \text{Bits} \vdash e : \text{TBits} \quad \lambda n.e[u/x] \text{ is numerical for all } u \]

\[ \lambda x . e[i(x)/n] \simeq \lambda x . e[B_1 i(x)/n] \text{ for all canonical } i \text{ such that } |i| < |p| \]

\[ \lambda x . e[nil/n] \simeq \lambda x . e[p(x)/n] \]

\[ \text{EQUIV} \]

\[ \text{TRANS-INDIST} \]

\[ \text{SUB} \]

\[ \text{H-IND} \]
An extendable set of lemmas:

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

- **RS-CONCAT**:

  $$bind \ x = rs(u) \ in \ bind \ y = rs(u') \ in \ val(x \cdot y) \equiv rs(u \cdot u')$$

- **RS-CUT**:

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

- **RS-COMMUT**:

  $$bind \ x = rs(u) \ in \ bind \ y = rs(v) \ in \ val(x \cdot y)
  \equiv bind \ x = rs(u) \ in \ bind \ y = rs(v) \ in \ val(y \cdot x)$$

- ....

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 $\lambda x. \text{bind } s = rs(x)$ in $\text{val}(G(s, p(s))) \simeq \lambda x. rs(p(x))$.
  - The proof follows the traditional hybrid technique, with the hybrid function $H \overset{\text{def}}{=} \lambda x. \lambda y. \lambda n. (y - n) \bullet G(x, n)$.

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

➢ The computational SLR
➢ The proof system for computational indistinguishability
➢ Game-based proofs in CSLR (joint work with D. Nowak)
➢ Conclusion
Game indistinguishability (joint work with D. Nowak)

- Proving computational indistinguishability is often hard, even in CSLR.
- Many cryptographers advocate game-based proofs:
  - Cryptosystems are described using games.
  - Crypto proofs are constructed as sequences of games.
  - Distances between neighboring games are negligible.
- Two CSLR programs $g_1, g_2$ of type $\square\text{Bits} \rightarrow (\square\text{Bits} \rightarrow \tau) \rightarrow \text{TBits}$ are game indistinguishable (written as $g_1 \simeq g_2$) 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 $\eta$,

\[
|\Pr[[g_1(\eta, \mathcal{A})] = 1] - \Pr[[g_2(\eta, \mathcal{A})] = 1]| < \frac{1}{|p(\eta)|}
\]
Security notions by game indistinguishability

✦ Next-bit unpredictability

\[
\lambda \eta. \lambda A. \quad \text{bind } s = rs(\eta) \text{ in } \\
\text{let } u = F(s) \text{ in } \\
\text{bind } b = A(\eta, \text{tail}(u)) \text{ in } \\
\text{val}(b \triangleq \text{head}(u)) \quad \cong \lambda \eta. \lambda A. \text{rand}
\]

✦ Semantic security

\[
\lambda \eta. \lambda (A_1, A_2). \quad \text{bind } (pk, sk) = KG(\eta) \text{ in } \\
\text{bind } (m_0, m_1) = A_1(\eta, pk) \text{ in } \\
\text{bind } b = \text{rand} \text{ in } \\
\text{bind } c = Enc(\eta, pk, m_b) \text{ in } \\
\text{bind } b' = A_2(\eta, pk, m_0, m_1, c) \text{ in } \\
\text{val}(b' \triangleq b) \quad \cong \lambda \eta. \lambda (A_1, A_2). \text{rand}
\]
Theorem: Computational indistinguishability implies game indistinguishability.

– The proof system of CSLR applies to game-based proofs.

We have verified in CSLR:

– Semantic security of El-Gamal encryption
– Next-bit unpredictability of Blum-Blum-Shub PSG
– These are checked based on their binary implementations.

We are working on:

– Hashed El-Gamal encryption in the random-oracle model.
– Non-game-based proofs of BBS.
– OAEP padding scheme.
– ... ...
Outline

➢ The computational SLR
➢ The proof system for computational indistinguishability
➢ Game-based proofs in CSLR
➢ Conclusion
Conclusion

✦ Contribution:
  – The first logic for cryptographic proofs with typing.
  – The first logic for cryptographic proofs without explicit bound.
  – The first verification based on the binary implementation of cryptographic schemes.

✦ Future work:
  – Automated proof checking.
  – More applications in cryptography.
  – Computational verification of protocols: higher order functions are already there.
  – Extension for reasoning about exact security.
  – Theoretical issues ...
Related work

- PPC by Mitchell et al. (2006): CCS-like language with bound replications; asymptotic bisimulation for computational indistinguishability.


- Automated verification game-based proofs:
  - Nowak (2007, 2008): shallow embedding (crypto schemes as distributions); implemented in Coq.
  - Barthe et al. (2009): deep embedding with an imperative language; relational Hoare logic; implemented in Coq.
  - Backes et al. (2008): functional language with references and events; implemented in Isabella/HOL; no real examples.