

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

Proofs in traditional cryptography

Proof. Let us define $K = 2^k$, $H_s = \lfloor \frac{p-1-s}{K} \rfloor$ for $s \in \llbracket 0, K-1 \rrbracket$. Let denote by e_p the following character of \mathbb{Z}_p : for all $y \in \mathbb{Z}_p$, $e_p(y) = e^{\frac{2\pi i y}{p}} \in \mathbb{C}^*$. The character e_p is an homomorphism from $(\mathbb{Z}_p, +)$ in (\mathbb{C}^*, \cdot) . Since

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

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

$$\begin{aligned} \Pr_{X \in G} [\text{lsb}_k(X) = s] &= \frac{1}{q} \times \left| \{(x, u) \in \llbracket 0, q-1 \rrbracket \times \llbracket 0, H_s \rrbracket \mid g^x = s + Ku \pmod p\} \right| \\ &= \frac{1}{qp} \times \sum_{x=0}^{q-1} \sum_{u=0}^{H_s} \sum_{a=0}^{p-1} e_p(a(g^x - s - Ku)). \end{aligned}$$

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 2Δ , is equal to:

$$\begin{aligned} &\sum_{s \in \{0,1\}^k} \left| \Pr_{X \in G} [\text{lsb}_k(X) = s] - 1/2^k \right| \\ &\leq \sum_{s \in \{0,1\}^k} \left| \frac{H_s + 1}{p} - \frac{1}{2^k} \right| + \sum_{s \in \{0,1\}^k} \frac{1}{qp} \sum_{a=1}^{p-1} \left| \left(\sum_{x=0}^{q-1} e_p(ag^x) \right) \left(\sum_{u=0}^{H_s} e_p(-aKu) \right) \right|. \end{aligned}$$

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

$$-\frac{1}{p} \leq -\frac{1+s}{Kp} \leq \left(1 + \left\lfloor \frac{p-1-s}{K} \right\rfloor \right) \frac{1}{p} - \frac{1}{K} \leq \frac{K - (1+s)}{Kp} \leq \frac{1}{p}.$$

- ❖ Cryptographers need a tool for checking proofs formally (and automatically).

Computational indistinguishability

- ❖ 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.



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**:

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

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

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

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

- Safe recursor: $\text{rec} : \mathbb{N} \rightarrow (\square(\mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \square(\mathbb{N}) \rightarrow \mathbb{N}$

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

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

The computational SLR (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.

Expressions:

e, e', \dots	$::=$	<code>nil</code>	empty bitstring
		<code>B₀ B₁</code>	bit successor
		<code>rec_τ</code>	safe recursor
		<code>rand</code>	oracle bit
		<code>val(<i>e</i>)</code>	trivial (deterministic) computations
		<code>bind <i>x</i> = <i>e</i> in <i>e</i>'</code>	sequential computations
		<code>...</code>	

- 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 terms.

- ❖ Typing rules:

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

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

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{val}(e) : \mathbb{T}\tau}$$

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

❖ 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.
- $\llbracket \text{rec} \rrbracket$ defines the safe recursion scheme.

❖ A probabilistic monad for interpreting probabilistic computations

$$\llbracket \mathsf{T}\tau \rrbracket = \llbracket \tau \rrbracket \rightarrow [0, 1]$$

$$\llbracket \text{rand} \rrbracket = \left\{ \left(0, \frac{1}{2}\right), \left(1, \frac{1}{2}\right) \right\}$$

$$\llbracket \text{val}(e) \rrbracket \rho = \{ (\llbracket e \rrbracket \rho, 1) \}$$

$$\llbracket \text{bind } x = e_1 \text{ in } e_2 \rrbracket \rho = \underline{\lambda} v. \sum_{v' \in \llbracket \tau \rrbracket} \llbracket e_2 \rrbracket \rho[x \mapsto v'](v) \times \llbracket e_1 \rrbracket \rho(v')$$

- The monad defined using measures [Ramsey & Pfeiffer '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 $\Box\text{Bits} \rightarrow \text{Bits}$ 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.

Cryptographic constructions in CSLR

- ❖ Goldreich and Micali's pseudorandom construction:

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

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

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

- ❖ Blum-Blum-Shub pseudorandom construction:

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

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 $\square \text{Bits} \rightarrow \text{Bits} \rightarrow \text{Bits} \rightarrow \text{Bits}$.

Outline

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

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 ,

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

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

Security notions by computational indistinguishability

- ❖ **Pseudorandomness**: a deterministic function G (of type $\square\text{Bits} \rightarrow \text{Bits}$) 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

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

- ❖ **Semantic security**:

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

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**.

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

$$\frac{e_1 \simeq e_2 \quad e_2 \simeq e_3}{e_1 \simeq e_3} \text{TRANS-INDIST}$$

$$\frac{x : {}^n \text{Bits}, y : {}^n \text{Bits} \vdash e : \text{TBits} \quad e_1 \simeq e_2}{\lambda x . \text{bind } y = e_1(x) \text{ in } e \simeq \lambda x . \text{bind } y = e_2(x) \text{ in } e} \text{SUB}$$

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

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 $H \stackrel{\text{def}}{=} \lambda x . \lambda y . \lambda n . (y - n) \bullet G(x, n)$.

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

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 $g_1 \cong g_2$) 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 ,

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

Security notions by game indistinguishability

❖ Next-bit unpredictability

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

❖ Semantic security

$$\begin{aligned} \lambda\eta. \lambda(\mathcal{A}_1, \mathcal{A}_2). \quad & \text{bind } (pk, sk) = \mathbf{KG}(\eta) \text{ in} \\ & \text{bind } (m_0, m_1) = \mathcal{A}_1(\eta, pk) \text{ in} \\ & \text{bind } b = \text{rand} \text{ in} \\ & \text{bind } c = \mathbf{Enc}(\eta, pk, m_b) \text{ in} \\ & \text{bind } b' = \mathcal{A}_2(\eta, pk, m_0, m_1, c) \text{ in} \\ & \text{val}(b' \stackrel{?}{=} b) \end{aligned} \cong \lambda\eta. \lambda(\mathcal{A}_1, \mathcal{A}_2). \text{rand}$$

Proving game indistinguishability

- ❖ 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.
- ❖ [Logics by Impagliazzo and Kapron \(2005\)](#): non-standard arithmetic model; explicit reasoning about probability.
- ❖ 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.
- ❖ [Blanchet's CryptoVerif \(2006\)](#): CCS-like language; automated generation of game sequences.