Formal Security Proofs with ICC

David Nowak

LIFL, CNRS & Lille 1 University

Joint work with Sylvain Heraud and Yu Zhang

Shonan Meeting on
Implicit Computational Complexity and applications:
Resource control, security, real-number computation

November 4–7, 2013
http://shonan.nii.ac.jp/seminar/033/

Outline

Formal security proofs in the computational model

Bellantoni-Cook

SLR

CSLR

Outline

Formal security proofs in the computational model

Bellantoni-Cook

SLR

CSLR

The problem with security proofs in cryptography

- Wrong proofs often find their way into top-level conferences.

  An infamous example: RSA-OAEP
  Industry-wide standard (PKCS#1 V2, IEEE P1363)

  - It was supposedly proved highly secure (Eurocrypt'94).
  - In fact, the proof had important holes (Crypto'01).
  - Those holes were finally(?) fixed by Pointcheval in 2005.

- Sometimes there are hidden assumption
  used in the proofs, but not stated in the theorem

- Some other a subtle point is underestimated by an author:
  "*One sees that...*", "*trivial*" or "*The reader may easily supply the details.*"

Formal security proofs within a proof assistant

$$\begin{array}{ccccc} \text{function} & \sqsubseteq & \text{algorithm} & \sqsubseteq & \text{program} \\ \text{true statement} & \sqsubseteq & \text{proof} & \sqsubseteq & \text{formal proof} \end{array}$$

- Formal security proofs in proof assistants:
  - In Coq [Affeldt et al. 2007, Nowak 2007, 2008, Barthe et al. 2009]

    Formal security proof for RSA-OAEP [Barthe et al., 2011]
  - In Isabelle [Berg, 2013]
- But none of the above frameworks deals with complexity.

The proof assistant Coq

- Based on a *kernel* which checks that:
  a given proof term p is really a proof of a given statement H.
- A tactic language (metalanguage) for building proofs incrementically
- Decision procedures and heuristics
- Notations, implicit parameters, coercions. . . .
- A standard library:
  arithmetic, analysis, polymorphic lists. . .
- The kernel is the only critical part:
  it will reject wrong proof terms.

The computational model

- Cf. Tuesday's talk by Bruce.
- Bruce's talk on Tuesday was on the computational soundness:

  Under which condition do we have the following?

  security in the symbolic model (Dolev-Yao) by logicians

  $\Downarrow$

  security in the computational model by cryptographers

- In this talk, I am talking about security proofs made directly in the computational model by cryptographers.

Security in the computational model

- An adversary is a function computable in probabilistic polynomial time (PPT),

  i.e., executable on a Turing machine extended with a read-only tape that has been filled with random bits, and working in worst-case polynomial time.

- A cryptographic scheme is a set of PPT functions.
  They are PPT:
  - for usability,
  - and also because they might be used by the adversary which has to be PPT.

- A security property is modeled as a probabilistic algorithm,
  i.e., a challenge that is to be solved by the adversary.

Example: ElGamal public-key encryption scheme

- ElGamal consists of the three following algorithms:

$$\text{keygen}() \;=\; x \xleftarrow{R} \mathbb{Z}_q \,;\; pk \leftarrow \gamma^x \,;\; sk \leftarrow x \,;\; \text{return } (sk, pk)$$

$$\text{encrypt}(pk, m) \;=\; y \xleftarrow{R} \mathbb{Z}_q \,;\; c \leftarrow (\gamma^y, pk^y * m) \,;\; \text{return } c$$

$$\text{decrypt}(sk, (c_1, c_2)) \;=\; m \leftarrow \frac{c_2}{c_1^{sk}} \,;\; \text{return } m$$

- Correctness is obvious: decryption indeed undoes encryption.
- Security is not so obvious.
  In fact, what do we mean by security?

Example: Semantic security

- **In English:** The challenger says to the adversary
  "*Give me two plaintexts; I will select one by flipping a coin, encrypt it, and give you the resulting cyphertext; You must then guess which of the two plaintexts I have encrypted.*"
- **As a probabilistic algorithm:**

$$
\begin{array}{l}
(pk, sk) \leftarrow \text{keygen}() \; ; \\
r \overset{R}{\leftarrow} R \; ; \\
(m_1, m_2) \leftarrow A_1(r, pk) \; ; \\
b \overset{R}{\leftarrow} \{1, 2\} \; ; \\
c \leftarrow \text{encrypt}(pk, m_b) \; ; \\
\hat{b} \leftarrow A_2(r, pk, c) \; ; \\
\textit{return} \;\; \hat{b} \overset{?}{=} b
\end{array}
$$

The cryptographic scheme is said "semantically secure" if for any adversary (A_1, A_2), the probability that this game returns true is negligibly close to $\frac{1}{2}$.

Security proofs in the computational model

- A security proof rely on a computational hypothesis, i.e., a problem that is believed not to be solvable in polynomial time.

  Example: Decisional Diffie-Hellman (DDH)
  No efficient algorithm can distinguish between triples of the form $(\gamma^x, \gamma^y, \gamma^{xy})$ and $(\gamma^x, \gamma^y, \gamma^z)$ where x, y and z are chosen randomly in $\mathbb{Z}_q$.

- Security proofs are done by contradiction:

  You assume an adversary A that can break the scheme (e.g., win the semantic security game).
  And, by using A, you build (usually, by game-hopping) another adversary that can break the computational hypothesis.

Why ICC?

- If you want to entirely formalize the proof in a proof assistant, you must formally prove that the newly built adversary is PPT.
- We do not want to count explicitly the number of steps in a precise execution model such as a Turing machine.
- We are interested in the complexity class, independently of the execution model.
- The right approach is ICC:

Outline

Formal security proofs in the computational model

Bellantoni-Cook

SLR

CSLR

The complexity class FP

- A function problem:
  Given an input x, output y such that $x\ R\ y$.
- A function problem is solvable in polynomial time if there exists a deterministic Turing machine M and a polynomial p such that:
  - On an input x, machine M halts after at most $p(|x|)$ steps, and
  - $M(x) = y$ iff $x\ R\ y$
- FP is the set of function problems that can be solved by a deterministic Turing machine in polynomial time.

Turing machines in Coq?

- It is not difficult to define Turing machines in Coq.
- But it is difficult to find a definition that will be usable.
- Even on paper, authors adapt the definition to their purpose
  - Moving head: $\{L, R\}$ or $\{L, R, N\}$?
  - One or more tapes?
  - ...
- We need an alternative definition of FP.

An alternative definition of FP

- FP by Cobham (1964):
  i. **Constant** 0
  ii. **Projection** $\pi_j^n(x_1, \ldots, x_n) = x_j$
  iii. **Successors** $s_i(x) = xi \quad$ for $i \in \{0, 1\}$
  iv. **smash** $2^{|x_1|.|x_2|}$
  v. **Recursion**
  $$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(yi, \overline{x}) &= h_i(y, \overline{x}, f(y, \overline{x})) \quad \text{for } yi \neq 0 \end{aligned}$$
  $|f(y, \overline{x})| <= |j(y, \overline{x})| \quad$ (`rec_bounded`)
  where g, h_0, h_1 and j are in this class
  vi. **Composition** $f(\overline{x}) = h(\overline{r}(\overline{x}))$
  where h and $\overline{r}$ are in this class

Cobham = FP

- This is exactly the class of functions computable in polynomial time on a deterministic Turing machine.
- The proof by Cobham uses a particular class of Turing machines but it is incidental. The results also holds with:
  - more than one tape,
  - multi-dimensional tapes,
  - instruction to erase the whole tape,
  - intruction to reset a scanning head.
  - ...
- We take Cobham's definition for FP.

A syntactic characterization of FP

- FP by Bellantoni and Cook (1992):
  i. **Constant** 0
  ii. **Projection** $\pi_j^{m,n}(x_1, \dots, x_m; x_{m+1}, \dots, x_{m+n}) = x_j$
  iii. **Successors** $s_i(; a) = ai \quad$ for $i \in \{0, 1\}$
  iv. **Predecessor** $p(; 0) = 0 \quad$ and $\quad p(; ai) = a$
  v. **Recursion** $f(0, \overline{x}; \overline{a}) = g(\overline{x}; \overline{a})$
     $f(yi, \overline{x}; \overline{a}) = h_i(y, \overline{x}; \overline{a}, f(y, \overline{x}; \overline{a})) \quad$ for $yi \neq 0$
     where g, h_0 and h_1 are in this class
  vi. **Composition** $f(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x};); \overline{t}(\overline{x}; \overline{a}))$
     where h, $\overline{r}$ and $\overline{t}$ are in this class
- There are two kind of variables separated by a semicolon:

$$f(\underbrace{x_1, \dots, x_n}_{\text{normal}}; \underbrace{a_1, \dots, a_s}_{\text{safe}})$$

Why Bellantoni-Cook is more convenient than Cobham

- When defining a recursive function f with Cobham, one has to exhibit a Cobham function j such that

$$|f(y, \overline{x})| <= |j(y, \overline{x})|$$

  In other words, there is a proof obligation.

- No such bound has to be proved with Bellantoni-Cook: This is a purely syntactic characterization of FP.

Bellantoni-Cook in Coq [Heraud and Nowak, 2011]

- Deep embedding of Cobham and Bellantoni-Cook classes
- Differences with the paper proof:
  - Fully constructive and tighter translations in both directions
  - We consider function on bitstrings instead of positive integers:
    As in cryptography, we distinguish bitstrings such as 010 and 00010.
- Integration with Certicrypt
- Although Bellantoni-Cook is a purely syntactic charaterization of polytime functions,
  it lacks features as a programming language:
  For example, for binary addition we would like to change the carry bit in the recursive call.

Outline

Formal security proofs in the computational model

Bellantoni-Cook

SLR

CSLR

SLR: Generalization to higher order

- SLR (Hofmann, 1997): a simply-typed lambda calculus with:
  - an S4 modality $\Box$, and
  - linear function spaces ($\multimap$).
- It generalizes Bellantoni and Cook's scheme to higher-order.
  - A function with m normal and n safe variables has type:

$$(\Box N)^m \rightarrow N^n \rightarrow N$$

  - It denotes a function f whose size is bounded:

$$|f(\overline{x}; \overline{a})| \leq P(|\overline{x}|) + max(|\overline{a}|)$$

- Linear functions are not needed to characterize polytime: They are an additional feature.
- Subtyping: $A \multimap B \quad <: \quad A \rightarrow B \quad <: \quad \Box A \rightarrow B$
- There is a type inference algorithm.

Examples of SLR functions and their inferred types

$$
\begin{array}{rcl}
\lambda x^{A}.x & : & A \multimap A \\
 & \vdots & \\
\lambda f^{A \to B}.\lambda x^{A}.f\ x & : & (A \to B) \multimap A \to B \\
 & \vdots & \\
\lambda f^{\Box A \to B}.\lambda x^{A}.f\ x & : & (\Box A \to B) \multimap \Box A \to B \\
 & \vdots & \\
\lambda f^{\Box A \to B}.\lambda g^{A \to A}.\lambda x^{A}.f(g\ x) & : & (\Box A \to B) \multimap \Box(A \to A) \to \Box A \to B
\end{array}
$$

Safe recursion in SLR

- SLR comes with a safe recursor:

$$\mathrm{saferec}_A \quad : \quad \Box N \to A \to (\Box N \to A \to A) \to A$$

- Its semantics is:

$$\begin{aligned} \mathrm{saferec}_A\; 0\; g\; h &= g \\ \mathrm{saferec}_A\; n\; g\; h &= h\; n\; (\mathrm{saferec}_A\; \lfloor n/2 \rfloor\; g\; h) \quad \text{when } n \neq 0 \end{aligned}$$

- Example: $sq\,x$ computes a value in the order of x^2:

$$sq : \Box N \to N \quad = \quad \lambda x^N.\mathrm{saferec}_N\; x\; 1\; (\lambda y^N.\lambda q^N.s_0(s_0 q))$$

- We can iterate sq: $\qquad \lambda x^N.sq\,(sq\,x) \quad : \quad \Box N \to N$
- But the following exponentially-growing function is ill-typed:

$$\lambda x^N.\mathrm{saferec}_N\; x\; 1\; (\lambda y^N.\lambda x^N.sq\; x)$$

Relation between Bellantoni and Cook's class and SLR

1. Define the category $\mathcal{C}$ of Bellantoni and Cook's functions.
   - Objects are pair of natural numbers
     (meant to be numbers of normal and safe arguments)
   - A morphisms from (m, n) to (m', n') is a pair of Bellantoni and Cook's functions $\left((f_1^{m,0}, \ldots, f_{m'}^{m,0}), (f_1^{m,n}, \ldots, f_{n'}^{m,n})\right)$
2. Embed $\mathcal{C}$ in the category $\widehat{\mathcal{C}}$ of presheaves over $\mathcal{C}$
   (i.e., the category of contravariant functors from $\mathcal{C}$ to Set).

   It is a standard application of Yoneda Lemma to embed first-order functions into a model of a higher-order typed language.

$$\begin{aligned}
\llbracket N \rrbracket &= \mathrm{Hom}_{\mathcal{C}}(-, (0,1)) \\
\llbracket A \to B \rrbracket = \llbracket A \multimap B \rrbracket &= \llbracket A \rrbracket \Rightarrow \llbracket B \rrbracket \\
\llbracket \Box A \to B \rrbracket = \llbracket \Box A \multimap B \rrbracket &= \Box \llbracket A \rrbracket \Rightarrow \llbracket B \rrbracket
\end{aligned}$$

3. **Theorem** (Hofmann) There is a bijection between the set of natural transformations from $\llbracket N \rrbracket^m \times \llbracket N \rrbracket^n$ to $\llbracket N \rrbracket$ and the set of (m, n)-ary functions in Bellantoni and Cook's class.

SLR in Coq

Bellantoni-Cook's class and its link to the complexity class FP	Yes (cf. Part 1)
SLR and its type system	Yes, but without linear types
Type inference	No
The category $\mathcal{C}$ of polytime functions	Yes
Embedding of $\mathcal{C}$ into the category $\widehat{\mathcal{C}}$ of presheaves	Yes
Set-theoretic semantics	Yes
Presheaf semantics	Yes
Logical relation between the two semantics	In progress, but Coq is too slow

Lessons learned from the formalization

- Category theory provides a very concise and abstract language for formal mathematics:

  Abstraction allows to factorize and thus reduce the development effort.

- When writing a statement in category theory, a lot of details are omitted because they can be recovered by the reader without ambiguity.
- Coq also can automatically recover the missing details thanks to mechanisms such as: implicit arguments, coercions...
- However, with category theory you need to push Coq to its limits:
  - Concise terms on screen can give rise to huge terms internally that will slow down Coq.
  - In some cases, the coercion mechanism needs type annotations.
  - You must keep track of universes to avoid universe inconsistencies.

Outline

Formal security proofs in the computational model

Bellantoni-Cook

SLR

CSLR

Adding a 0,1-valued oracle

- (Mitchell et al., 1998) extend SLR with a 0,1-valued oracle.
  Another standard categorical technique is used:
  The Kleisli construction
- OSLR characterizes probabilistic polytime functions.
- The oracle is a kind of side-effect:
  The resulting value depends of the evaluation strategy.
- It makes difficult to build a logic upon the language.
- A standard solution used by (Zhang, 2009) is to hide the side-effect with a monadic type.

CSLR

- CSLR (Zhang, 2009) extend OSLR with monadic types:

$$\tau ::= \cdots \mid \mathsf{T}\tau$$

  They distinguish at type level between deterministic and probabilistic computations.

- The type N is replaced by the type Bits for bitstrings.
  - 0 and 00 (for example) are different bitstrings in CLSR but were identified to the nu mber 0 in SLR.

- Expressions are extended with probabilistic computations:

$$e ::= \cdots \mid \mathsf{rand} \mid \mathsf{return}(e) \mid x \stackrel{\$}{\leftarrow} e_1;\ e_2$$

- It allows to build a logic for reasoning about computational indistinguishability.

An example of CSLR function

- To ease the reading of CSLR terms, we use syntactic sugar
  - In particular, a term F defined recursively by $\lambda n\,.\,\mathtt{rec}_\tau(e_1, e_2, n)$ is written:

$$F \stackrel{\text{def}}{=} \lambda n\,.\,\mathtt{if}\ n \stackrel{?}{=} \mathtt{nil}\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2(n, F(\textbf{\textit{tail}}(n))),$$

- The random bitstring generation:

$$\begin{aligned}\textbf{\textit{rs}} \stackrel{\text{def}}{=} \lambda n\,.\,&\mathtt{if}\ (n \stackrel{?}{=} \mathtt{nil})\\ &\mathtt{then}\ \mathtt{return}(\mathtt{nil})\\ &\mathtt{else}\ b \stackrel{\$}{\leftarrow} \mathtt{rand};\ u \stackrel{\$}{\leftarrow} \textbf{\textit{rs}}(\textbf{\textit{tail}}(n));\ \mathtt{return}(b \bullet u)\end{aligned}$$

- Input: a bitstring
  Output: a random bitstring of the same length
- One can check that $\vdash \textbf{\textit{rs}} : \Box\mathrm{Bits} \to \mathrm{TBits}$

Pseudo-uniform sampling

- In theoretical proofs, arbitrary uniform sampling are used.
  (for example, $x \in_R \mathbb{Z}_n^*$)
  - But in practice, computers are based on binary digits:

    The cardinal of a uniform distribution has to be a power of 2.
  - The complexity class PPT is defined with probabilistic Turing machines.

    But probabilistic Turing machines deal with random bits only.

- Pseudo-uniform sampling in CSLR:

$$
\boldsymbol{zrand} \stackrel{\text{def}}{=} \lambda n \,.\, \lambda t \,.\, \begin{array}[t]{l} \mathtt{if}\ t \stackrel{?}{=} \mathtt{nil} \\ \mathtt{then}\ \mathtt{return}(0^{|n|}) \\ \mathtt{else}\ v \stackrel{\$}{\leftarrow} \boldsymbol{rs}(n); \\ \qquad \mathtt{if}\ v \geq n \\ \qquad \mathtt{then}\ \boldsymbol{zrand}(n, \boldsymbol{tail}(t)) \\ \qquad \mathtt{else}\ \mathtt{return}(v) \end{array}
$$

Tries to sample a value between 0 and n.
After a timeout $|t|$, it returns the default value $0^{|n|}$.

Indistinguishability

- *Two CSLR terms f_1 and f_2 are* **computationally indistinguishable** *(written as $f_1 \simeq f_2$) if for every term $\mathcal{A}$ such that $\vdash \mathcal{A} : \Box\mathsf{Bits} \to \tau \to \mathsf{TBits}$ and every positive polynomial P, there exists some $N \in \mathbb{N}$ such that for all bitstring η with $|\eta| \geq N$*

$$|\mathbf{Pr}[\llbracket \mathcal{A}(\eta, f_1(\eta)) \rrbracket \rightsquigarrow 1] - \mathbf{Pr}[\llbracket \mathcal{A}(\eta, f_2(\eta)) \rrbracket \rightsquigarrow 1]| < \tfrac{1}{P(|\eta|)}$$

$$\Downarrow$$

- *Two CSLR terms g_1 and g_2 are* **game indistinguishable** *(written as $g_1 \approx g_2$) if for every term $\mathcal{A}$ such that $\vdash \mathcal{A} : \Box\mathsf{Bits} \to \mathsf{T}\tau$, and every positive polynomial P, there exists some $N \in \mathbb{N}$ such that for all bitstring η with $|\eta| \geq N$,*

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

Uniform sampling

We also show that the standard practice of cryptographers, ignoring that polynomial-time Turing machines cannot generate all uniform distributions, is actually sound.

- $\mathrm{CSLR}^{\$}$ extends CSLR with a uniform sampling primitive `sample` of type Bits $\multimap$ TBits.
- We prove that we can freely replace the approximate uniform sampling ***zrand*** by the truly uniform sampling `sample` or vice versa in sampling-based CSLR programs, without affecting the computational indistinguishability.

Superpolynomial constants

- Game-hopping does not preclude the possibility of introducing games that perform superpolynomial-time computations.
- They are just idealized constructions that are used to define security notions but are not meant to make their way into implementations.
- $\mathsf{CSLR}^{\$}_{\pi}$ extends $\mathsf{CSLR}^{\$}$ with a set π of superpolynomial-time primitives.

Conclusions

- In Coq, we currently have:
  - a formalization of Bellantoni-Cook, and
  - an almost finished formalization of SLR.

- We propose an extension of CSLR into $\mathrm{CSLR}^{\$}_{\pi}$ [Nowak and Zhang, 2013] that allows for convenient formalization of game-hopping security proofs taking into account complexity issues.

Thank you!