CertiCrypt: formal certification of code-based cryptographic proofs

Gilles Barthe Benjamin Grégoire Santiago Zanella Béguelin
Romain Janvier Féderico Olmedo

IMDEA Software
INRIA Sophia Antipolis
INRIA-Microsoft Research Joint Centre
National University of Rosario

15.07.2008

What's wrong with cryptographic proofs?

Increasing complexity in cryptographic proofs

+

Unmanageable numbers of them appearing in articles

+

No one willing to carefully verify long handmade proofs

Subtle errors in published proofs

What's wrong with cryptographic proofs?

Increasing complexity in cryptographic proofs

+

Unmanageable numbers of them appearing in articles

+

No one willing to carefully verify long handmade proofs

Subtle errors in published proofs

What's wrong with cryptographic proofs?

Increasing complexity in cryptographic proofs

+

Unmanageable numbers of them appearing in articles

+

No one willing to carefully verify long handmade proofs

Subtle errors in published proofs

What's wrong with cryptographic proofs?

Increasing complexity in cryptographic proofs

+

Unmanageable numbers of them appearing in articles

+

No one willing to carefully verify long handmade proofs

Subtle errors in published proofs

From provable cryptography to proved provable cryptography

Provable security

- State security goals precisely
- Make security hypotheses explicit
- Carry rigorous proofs

- State security goals and hypotheses formally (in a fully specified formalism)
- Develop tool supported methods for building or checking proofs

Proposal: game-based proofs
(Not a universal point of view)

Game-based proofs

- Describe security of system as a game
  - Game as probabilistic program
  - Security as upper bound on the adversary's advantage
  - Security assumptions as games
- Transform game stepwise $G, E, p \longrightarrow G', E', p'$
  - p' should be suitably related to p
  - E and E' may be distinct events
    (e.g. adversary winning vs failure event)
- Provide upper bound for probability in the final game

Caveats

- Game hopping is only part of the story
- Many (complex) side results must be established
  (PPT, probability, etc)
- Ad hoc reasoning might be required

IND-CPA

$$\textbf{Game}_0 = \begin{array}{l} (pk, sk) \leftarrow KG(\eta); \\ M_1, M_2 \leftarrow A(pk); \\ b \stackrel{\$}{\leftarrow} \{0,1\}; \\ \text{if } b \text{ then } M_b \leftarrow M_1 \text{ else } M_b \leftarrow M_2; \\ Y' \leftarrow Enc(sk, M_b); \\ b' \leftarrow A'(Y') \end{array}$$

- Asymptotic security: show that $|Pr_{\textbf{Game}_0}[b = b'] - \frac{1}{2}|$ is negligible in k
- Exact security: provide L such that $|Pr_{\textbf{Game}_0}[b = b'] - \frac{1}{2}| \leq L(k)$

Semantic security of ElGamal

- Key generation: $\mathcal{KG}() \stackrel{\triangle}{=} x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$ return (x, g^x)
- Encryption: $\mathsf{Enc}(\alpha, m) \stackrel{\triangle}{=} y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$ return $(g^y, \alpha^y \times m)$

ElGamal is INDCPA secure under DDH

Decisional Diffie-Hellman (DDH) assumption

Let G be a cyclic group of order q, let g be a generator of G.

$$\begin{array}{rcl} DDH_0 & = & x \stackrel{\$}{\leftarrow} [0..q-1];\ y \stackrel{\$}{\leftarrow} [0..q-1]; \\ & & b \leftarrow A(g^x, g^y, g^{x*y}); \end{array}$$

$$\begin{array}{rcl} DDH_1 & = & x \stackrel{\$}{\leftarrow} [0..q-1];\ y \stackrel{\$}{\leftarrow} [0..q-1]; z \stackrel{\$}{\leftarrow} [0..q-1]; \\ & & b \leftarrow A(g^x, g^y, g^z); \end{array}$$

For all PPT adversaries, $|\Pr_{DDH_0}[b = 1] - \Pr_{DDH_1}[b = 1]|$ is negligible in k.

Game hopping

Game ElGamal :
$(x, \alpha) \leftarrow \mathcal{KG};$
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$(\beta, \zeta) \leftarrow \mathrm{Enc}(\alpha, m_b);$
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \zeta)$
$d \leftarrow b = b'$

Game ElGamal$_0$:
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$\zeta \leftarrow g^{xy} \times m_b;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$d \leftarrow b = b'$

Game DDH$_0$:
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$
Adversary $\mathcal{B}(\alpha, \beta, \gamma)$:
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b);$
return $b = b'$

Proof steps

Game hopping

Game ElGamal :
$(x, \alpha) \leftarrow \mathcal{KG}$;
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
$b \stackrel{\$}{\leftarrow} \{0, 1\}$;
$(\beta, \zeta) \leftarrow \text{Enc}(\alpha, m_b)$;
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \zeta)$
$d \leftarrow b = b'$

Game ElGamal_0 :
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q$; $y \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
$(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
$b \stackrel{\$}{\leftarrow} \{0, 1\}$;
$\zeta \leftarrow g^{xy} \times m_b$;
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta)$;
$d \leftarrow b = b'$

Game DDH_0 :
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
$y \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
$d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$
Adversary $\mathcal{B}(\alpha, \beta, \gamma)$:
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
$b \stackrel{\$}{\leftarrow} \{0, 1\}$;
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b)$;
return $b = b'$

Proof steps

```
inline_l KG.
inline_l Enc.
ep.
deadcode.
swap.
eqobs_in.
```

Game hopping

Game ElGamal :
$(x, \alpha) \leftarrow \mathcal{KG}$;
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
$b \stackrel{\$}{\leftarrow} \{0,1\}$;
$(\beta, \zeta) \leftarrow \mathrm{Enc}(\alpha, m_b)$;
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \zeta)$
$d \leftarrow b = b'$

Game $\mathrm{ElGamal}_0$:
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q$; $y \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
$(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
$b \stackrel{\$}{\leftarrow} \{0,1\}$;
$\zeta \leftarrow g^{xy} \times m_b$;
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta)$;
$d \leftarrow b = b'$

Game DDH_0 :
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
$y \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
$d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$
Adversary $\mathcal{B}(\alpha, \beta, \gamma)$:
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
$b \stackrel{\$}{\leftarrow} \{0,1\}$;
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b)$;
return $b = b'$

Proof steps

```
inline_r B.
ep.
deadcode.
eqobs_in.
```

Game hopping

Game ElGamal_2 :

$x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$z \xleftarrow{\$} \mathbb{Z}_q;\ \zeta \leftarrow g^z;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$b \xleftarrow{\$} \{0,1\};$
$d \leftarrow b = b'$

Game ElGamal_1 :

$x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$b \xleftarrow{\$} \{0,1\};$
$z \xleftarrow{\$} \mathbb{Z}_q;\ \zeta \leftarrow g^z \times m_b;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$d \leftarrow b = b'$

Game DDH_1 :

$x \xleftarrow{\$} \mathbb{Z}_q;$
$y \xleftarrow{\$} \mathbb{Z}_q;$
$z \xleftarrow{\$} \mathbb{Z}_q;$
$d \leftarrow \mathcal{B}(g^x, g^y, g^z)$

Adversary $\mathcal{B}(\alpha, \beta, \gamma)$:

$(m_0, m_1) \leftarrow \mathcal{A}(\alpha);$
$b \xleftarrow{\$} \{0,1\};$
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b);$
return $b = b'$

Proof steps

Game hopping

Game $\mathsf{ElGamal}_2$:

$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$z \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ \zeta \leftarrow g^z;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$d \leftarrow b = b'$

Game $\mathsf{ElGamal}_1$:

$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$z \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ \zeta \leftarrow g^z \times m_b;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$d \leftarrow b = b'$

Game DDH_1 :

$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$z \stackrel{\$}{\leftarrow} \mathbb{Z}_q;$
$d \leftarrow \mathcal{B}(g^x, g^y, g^z)$

Adversary $\mathcal{B}(\alpha, \beta, \gamma)$:

$(m_0, m_1) \leftarrow \mathcal{A}(\alpha);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b);$
return $b = b'$

Proof steps

```
swap.
eqobs_hd 4.
eqobs_tl 2.
apply mult_pad.
```

Game hopping

Game ElGamal$_2$:
$x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$z \xleftarrow{\$} \mathbb{Z}_q;\ \zeta \leftarrow g^z;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$b \xleftarrow{\$} \{0,1\};$
$d \leftarrow b = b'$

Game ElGamal$_1$:
$x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$b \xleftarrow{\$} \{0,1\};$
$z \xleftarrow{\$} \mathbb{Z}_q;\ \zeta \leftarrow g^z \times m_b;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$d \leftarrow b = b'$

Game DDH$_1$:
$x \xleftarrow{\$} \mathbb{Z}_q;$
$y \xleftarrow{\$} \mathbb{Z}_q;$
$z \xleftarrow{\$} \mathbb{Z}_q;$
$d \leftarrow \mathcal{B}(g^x, g^y, g^z)$
Adversary $\mathcal{B}(\alpha, \beta, \gamma)$:
$(m_0, m_1) \leftarrow \mathcal{A}(\alpha);$
$b \xleftarrow{\$} \{0,1\};$
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b);$
return $b = b'$

Proof steps

```
inline_r B.
ep.
deadcode.
swap.
eqobs_in.
```

Wrapping up

- Equational proof

$$
\begin{aligned}
|\Pr_{\mathsf{ElGamal}}(b = b') - \tfrac{1}{2}| &= |\Pr_{\mathsf{ElGamal}_0}(d) - \tfrac{1}{2}| \\
&= |\Pr_{\mathsf{DDH}_0}(d) - \tfrac{1}{2}| \\
&= |\Pr_{\mathsf{DDH}_0}(d) - \Pr_{\mathsf{ElGamal}_2}(d)| \\
&= |\Pr_{\mathsf{DDH}_0}(d) - \Pr_{\mathsf{ElGamal}_1}(d)| \\
&= |\Pr_{\mathsf{DDH}_0}(d) - \Pr_{\mathsf{DDH}_1}(d)|
\end{aligned}
$$

- Needs proof that DDH is correctly applied!

Random oracle

$$
\begin{aligned}
G :\quad & \{0,1\}^p \to \{0,1\}^p \\
G(R) \triangleq\quad & \text{if } R \notin L \text{ then} \\
& \quad r \xleftarrow{\$} \{0,1\}^k; \\
& \quad L \leftarrow (R, r) :: L \\
& \text{else } r \leftarrow L[R] \\
& \textbf{return } r
\end{aligned}
$$

The OAEP padding scheme

- A one-way permutation function $f : \{0,1\}^k \rightarrow \{0,1\}^k$
- Two hash functions:
  - $G : \{0,1\}^p \rightarrow \{0,1\}^{k-p}$
  - $H : \{0,1\}^{k-p} \rightarrow \{0,1\}^p$
- Encryption:

$$Enc(M) \triangleq \quad \begin{array}{l} R \stackrel{\$}{\leftarrow} \{0,1\}^p; \\ S \leftarrow G(R) \oplus M; \\ T \leftarrow H(S) \oplus R; \\ Y \leftarrow f(S \| T); \\ \textbf{return } Y \end{array}$$

Exact security of OAEP

- Proved in Coq (2,500 lines):

$$|Pr_{\mathbf{Game}_0}[b = b'] - \frac{1}{2}| \leq Pr_{I,f} + \frac{q_G}{2^p}$$

  where $Pr_{I,f}$ is the probability of an adversary I to invert f on a random element

- Improves over Bellare and Rogaway:

$$|Pr_{\mathbf{Game}_0}[b = b'] - \frac{1}{2}| \leq Pr_{I,f} + \frac{2q_G}{2^p} + \frac{q_H}{2^{k-p}}$$

- ... but we should really prove IND-CCA!

Goals and rationale

Goal

Build a certified tool for checking game-playing proofs, on top of a general purpose proof assistant (Coq)

- Security goals, properties and hypotheses are explicit
- Game hopping and side conditions are justified in a unified formalism
- The tool provides independently checkable certificates

Not primary goals

- Discovering the sequence of games, interface
- Protocols

Building blocks

- Probability library (Paulin and Audebaud)
- Programming language
- Semantics
  - Execution
  - Complexity and termination
- Security definitions
- Tools:
  - Observational equivalence and relational logic
  - Program transformations
  - Game-based lemmas
- Examples

Language

Probabilistic, procedural *while* language

$$
\begin{array}{llcl}
\text{Expressions} & e & ::= & x \mid op\ \vec{e} \\
\text{Instructions} & i & ::= & x \leftarrow e \mid x \stackrel{\$}{\leftarrow} d \\
 & & \mid & \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \mid \mathsf{while}\ e\ \mathsf{do}\ s \\
 & & \mid & x \leftarrow f(e_1, \ldots, e_n) \\
\text{Statements} & s & ::= & [] \mid i; s \\
\text{Environments} & E & : & f \mapsto \vec{x} * s * e
\end{array}
$$

- Formalism that is already used by cryptographers (but we have while loops)
- Syntax is extensible with new operators and types
- Extensive use of new module system (Coq V8.2)

Games a programs

- Game and oracles are described as procedures
- Adversaries are uninterpreted procedures
- Must also specify:
  - which variables can be accessed/modified
  - which procedures can be called
    (how many times, under which restrictions)

Each definition of procedure f includes termination flag, and

- variables O
- variables I that must coincide on entry for result and final memories (restricted to O) to coincide on any two runs of f

Functions to compute automatically the required information.
Possibly instrument code, e.g. to count number of calls.

Semantics: dependent types at work

- Type system
  - Avoids partial semantics of expressions
  - Enforces size constraints (e.g. length of bitstrings)
- Embedded in the semantics using dependent types

$$\begin{array}{llll} \text{Values} & v & ::= & n \mid b \mid bs \\ \text{Expressions} & e & ::= & x \mid v \mid e_1 + e_2 \mid e_1 \mathbin{\&\&} e_2 \mid b_1 +\!\!+ b_2 \end{array}$$

Formalisation

$$\begin{array}{l} \text{Inductive type} : \text{Type} := \ldots \\ \text{Inductive var} \ \ : \text{type} \rightarrow \text{Type} := \ldots \\ \text{Inductive expr} \ \ : \text{type} \rightarrow \text{Type} := \\ \quad \mid \text{Evar} : \forall t,\ \text{var}\ t \rightarrow \text{expr}\ t \\ \quad \mid \text{Eop} : \forall op,\ \text{dlist expr}\ (\text{Op.targs}\ op) \rightarrow \text{expr}\ (\text{Op.tres}\ op). \end{array}$$

Security parameter: dependent types at work

- Security parameter must be explicit in model.
- We parametrize the semantics by the security parameter

$$\text{Definition interp} : \text{nat} \rightarrow \text{type} \rightarrow \text{Type} := \ldots$$

$$\text{Parameter Mem.t} \;\; : \text{nat} \rightarrow \text{Type}.$$

$$\text{Parameter get} : \forall k,\; \text{Mem.t}\; k \rightarrow \forall t,\; \text{var}\; t \rightarrow \text{interp}\; k\; t.$$

$$\begin{array}{l} \text{Fixpoint eval}\; k\; t\; (e : \text{expr}\; t)\; (m : \text{Mem.t}\; k) : \text{interp}\; k\; t := \\ \quad \text{match}\; e\; \text{with} \\ \quad \mid \text{Evar}\; t\; x \Rightarrow \text{get}\; k\; m\; t\; x \\ \quad \mid \text{Eop}\; op\; args \Rightarrow \text{dapp}\; (\text{interpop}\; op)\; (\text{dmap}\; (\text{eval}\; k)\; args) \\ \quad \text{end.} \end{array}$$

Execution

- Given a command, an initial state, returns the distribution for final states:

$$\Pr : \mathcal{C} \to \mathcal{S} \to \mathcal{S} \to [0,1]$$

- Given a command, an initial state, and an expectation function, returns the probability for final states:

$$[\![\cdot]\!] : \mathcal{C} \to \mathcal{S} \to (\mathcal{S} \to [0,1]) \to [0,1]$$

- Both semantics are formally related.

$$[\![c]\!]\ \sigma\ f = \sum_{\sigma' \in \mathcal{S}} f(\sigma')\ \Pr[\langle c, \sigma\rangle \downarrow \sigma']$$

We formalize the second.

Small-step semantics, formally

- Notation: $\mathcal{D}_A = (A \to [0,1]) \to [0,1]$
- States are frame-based
- Each frame is a record: local memory, statement, etc
- Small-step semantics $[\![\cdot]\!]_1 : \mathcal{C} \to \mathcal{S} \to \mathcal{D}_{\mathcal{S}}$ defined by case analysis on the instruction to be executed
- Evaluation semantics $[\![\cdot]\!] : \mathcal{C} \to \mathcal{M} \to \mathcal{D}_{\mathcal{M}}$ defined as least upper bound of the n-unfold $[\![\cdot]\!]_n$ of $[\![\cdot]\!]_1$:

$$[\![c]\!]\ \mu\ f = \text{lub}\ (\lambda n \cdot [\![c]\!]_n\ \mu\ f!)$$

where $f!$ is the restriction of f to final states.

Example

- Probability of event E:

$$\Pr_{c,\sigma}[E] = [\![c]\!]\ \sigma\ 1_E$$

- Example:

$$\begin{array}{rcl}
\Pr_{x \overset{\$}{\leftarrow} [0,1], \sigma}[x = 0] & = & [\![x \overset{\$}{\leftarrow} [0,1]]\!]\ \sigma\ 1_{x=0} \\
& = & \frac{1}{2}.(1_{x=0}\ \sigma\{x := 0\}) + \frac{1}{2}.(1_{x=0}\ \sigma\{x := 1\}) \\
& = & \frac{1}{2}.1 + \frac{1}{2}.0 \\
& = & \frac{1}{2}
\end{array}$$

Termination

Program is lossless iff $\Pr_{c,\sigma}[\mathrm{True}] = 1$

- Semantic definition
- Rules for constructs (except loops)
- Tactic for generating proof of losslessness for programs without loops

PPT complexity

- Semantics of programs instrumented with cost monad:

$$[\![\cdot]\!] : \mathcal{C} \to (\mathcal{S} \times \mathbb{N}) \to (\mathcal{S} \times \mathbb{N} \to [0,1]) \to [0,1]$$

- A state $(m, n)_k$ is (p, q) bounded if $p(k)$ bounds the size of values in the memory and $n \leq q(k)$ (p and q be polynomials on the security parameter)
- A program c is strict PPT iff it is lossless and

$$\begin{array}{ll} \exists\ F,\ G \cdot & \forall\ (d : \mathcal{D}_{\mathcal{S}\times\mathbb{N}}),\ (p,\ q : \mathbb{N}[x]) \\ & \text{range (bounded } p\ q)\ d \Rightarrow \\ & \text{range (bounded } (F\ p)\ (q + G\ p))\ (\text{bind } d\ [\![c]\!]) \end{array}$$

- Semantic definition, together with rules for constructs
- Tactic for generating proof of PPT for programs without loops

Relational Hoare Logic

- Deterministic setting: $c_1 \; P \simeq Q \; c_2$ iff

$$\forall m_1 \; m_2, \; P \; m_1 \; m_2 \rightarrow Q \; [\![c_1]\!]_{m_1} \; [\![c_2]\!]_{m_2}$$

  where P and Q are relations on memories
- Probabilistic setting: $c_1 \; P \simeq Q \; c_2$ iff

$$\forall m_1 \; m_2, \; P \; m_1 \; m_2 \rightarrow \mathrm{lift} \; Q \; [\![c_1]\!]_{m_1} \; [\![c_2]\!]_{m_2}$$

  (Remark: in pRHL P and Q still are relations on memories. Working on an extension to distributions.)
- Question: how do we lift Q?

Probabilistic Relational Hoare Logic

- Range of distribution

$$\text{range } A\ P\ (d : \mathcal{D}_A) := \forall f, (\forall a,\ P\ a \rightarrow 0 = f\ a) \rightarrow 0 = d\ f$$

- Lifting relation

$$\begin{aligned}\text{lift } A\ B\ R\ (d_1 : \mathcal{D}_A)\ (d_2 : \mathcal{D}_B) := \exists (d : \mathcal{D}_{A*B}),\\ \pi_1(d) = d_1 \wedge \pi_2(d) = d_2\ \wedge \text{range } (A * B)\ R\ d\end{aligned}$$

- Observational equivalence

$$c_1\ P \simeq Q\ c_2 := \forall m_1\ m_2,\ P\ m_1\ m_2 \rightarrow \text{lift } Q\ [\![c_1]\!]_{m_1}\ [\![c_2]\!]_{m_2}$$

If f and g do not distinguish memories related by Q, i.e.

$$\forall m_1\ m_2, Q\ m_1\ m_2 \rightarrow f\ m_1 = g\ m_2$$

then

$$\forall m_1\ m_2, P\ m_1\ m_2 \rightarrow [\![c_1]\!]_{m_1} f = [\![c_2]\!]_{m_2} g$$

Selected rules

$$\frac{c_1 \;\; P \simeq Q \;\; c_2 \qquad P' \Rightarrow P}{c_1 \;\; P' \simeq Q \;\; c_2} \qquad\qquad \frac{c_1 \;\; P \simeq Q \;\; c_2 \qquad Q \Rightarrow Q'}{c_1 \;\; P \simeq Q' \;\; c_2}$$

$$\frac{Q' := \lambda m_1 \;\; m_2, Q \;\; m_1\{x_1 := [\![e_1]\!]_{m_1}\} \;\; m_2\{x_2 := [\![e_2]\!]_{m_2}\}}{x_1 \leftarrow e_1 \;\; Q' \simeq Q \;\; x_2 \leftarrow e_2}$$

$$\frac{\begin{array}{c} Q' := \lambda k \;\; m_1 \;\; m_2, \textbf{permut\_support} \;\; f \;\; d_1 \;\; d_2 \;\; k \;\; m_1 \;\; m_2 \wedge \\ \forall v \in [\![d_2]\!]_{m_2}, Q \;\; m_1\{x := f \;\; k \;\; m_1 \;\; m_2 \;\; v\} \;\; m_2\{x := v\} \end{array}}{x \stackrel{\$}{\leftarrow} d_1 \;\; Q' \simeq Q \;\; x \stackrel{\$}{\leftarrow} d_2}$$

$$\frac{c_1 \;\; P \simeq Q \;\; c_1' \qquad c_2 \;\; Q \simeq R \;\; c_2'}{c_1; c_2 \;\; P \simeq R \;\; c_1'; c_2'}$$

$$\frac{c_1 \;\; P|_e \simeq Q \;\; c_1' \qquad c_2 \;\; P|_{\neg e} \simeq Q \;\; c_2' \qquad [\![e]\!] \simeq_P [\![e']\!]}{\text{if } e \text{ then } c_1 \text{ else } c_2 \;\; P \simeq Q \;\; \text{if } e' \text{ then } c_1' \text{ else } c_2'}$$

Lifting of a relation

Justifying our definition

$$x \xleftarrow{\$} \{0,1\} \; True \simeq\!=_{\{x\}} x \xleftarrow{\$} \{0,1\}$$

- With product distribution, equivalence would fail, because pairs $(0,1)$ and $(1,0)$ violate the postcondition and have a non-null probability.
- With our definition, we can choose the distribution that gives probability $1/2$ to $(0,0)$ and $1/2$ to $(1,1)$. Equivalence holds.

Beware

$$x \xleftarrow{\$} \{0,1\} \; True \simeq\!\neq_{\{x\}} x \xleftarrow{\$} \{0,1\}$$

- With our definition, we can choose the distribution that gives probability $1/2$ to $(0,1)$ and $1/2$ to $(1,0)$. Equivalence holds.
- Intuitively, the relation $=_{\{x\}}$ cannot distinguish two executions of the command.

Fundamental properties of pRHL

- Termination sensitivity

$$\left.\begin{array}{c} \models G_1 \sim G_2 : \Psi \Rightarrow \Phi \\ m_1 \ \Psi \ m_2 \end{array}\right\} \Rightarrow [\![G_1]\!] \ m_1 \ \mathbb{1} = [\![G_2]\!] \ m_2 \ \mathbb{1}$$

- Equivalence implies inseparability

$$\left.\begin{array}{c} \models G_1 \sim G_2 : \Psi \Rightarrow \Phi \\ f =_\Phi g \\ m_1 \ \Psi \ m_2 \end{array}\right\} \Rightarrow [\![G_1]\!] \ m_1 \ f = [\![G_2]\!] \ m_2 \ g$$

where

$$f =_\Phi g \stackrel{\triangle}{=} \forall \ m_1 \ m_2. \ m_1 \ \Phi \ m_2 \Rightarrow f(m_1) = g(m_2)$$

- Variant

$$\left.\begin{array}{c} \models G_1 \sim G_2 : \Psi \Rightarrow \Phi \\ f \leq_\Phi g \\ m_1 \ \Psi \ m_2 \end{array}\right\} \Rightarrow [\![G_1]\!] \ m_1 \ f \leq [\![G_2]\!] \ m_2 \ g$$

Specialize relational Hoare logic to local equality of memories

$$\simeq_O^I \;=\; =_I \simeq =_O$$

- Code motion: $I = fv(e_1) \cup fv(e_2)$ and $x \notin fv(e_2)$ and $y \notin fv(e_1)$

$$x \leftarrow e_1; y \leftarrow e_2 \simeq_{\{x,y\}}^I y \leftarrow e_2; x \leftarrow e_1$$

- Dead code:

$$x \leftarrow 3; y \stackrel{\$}{\leftarrow} [0,1] \simeq_{\{x\}}^{\emptyset} x \leftarrow 3$$

Question

Do we have

$$x \leftarrow 3; y \leftarrow f(1) \simeq_{\{x\}}^{\emptyset} x \leftarrow 3$$

Algebraic examples

- Let G be a cyclic group of order q, g a generator, and m an element of G

$$z \xleftarrow{\$} [0..q-1]; w \leftarrow g^z * m \simeq^I_O z \xleftarrow{\$} [0..q-1]; w \leftarrow g^z$$

I	O	
$\{q, g\}$	$\{z\}$	OK, dead code
$\{q, g\}$	$\{w\}$	OK
$\{q, g\}$	$\{z, w\}$	KO

- Let k be a fixed constant

$$z \xleftarrow{\$} \{0,1\}^k; w \leftarrow z \oplus c \simeq^{\{c\}}_{\{c,z\}} w \xleftarrow{\$} \{0,1\}^k; z \leftarrow w \oplus c$$

Automating proofs

Goal

Provide proof support for goals of the form

$$c_1 \; P \simeq Q \; c_2 \qquad\qquad c_1 \simeq^I_O c_2$$

- Many transformations correspond to program optimizations
- Transformations are programmed and proved correct in Coq (proof by reflection)
- We have developed a set of tactics to apply automatically these transformations

Main transformations

- Dependency analysis for showing $c \simeq^I_O c$

$$c \simeq^?_O c \qquad\qquad c \simeq^I_? c$$

  (strong relation with information flow analysis)
- Dead code (wrt a set O of output variables)

$$\mathrm{dead\_code}\ c\ O = (I, c') \to c \simeq^I_O c'$$

  (acts as an aggressive slicing algorithm)
- Code motion

$$\forall c_1\ c_2, \mathrm{swap}\ c_1\ c_2 = \mathit{true} \to \forall P\ Q, c_1\ P \simeq Q\ c_2$$

- Constant propagation
- Expression propagation (def. unfolding+partial eval.)
- Inlining

Fundamental lemma

If two programs **Game**$_1$ and **Game**$_2$ are *equivalent* up to a failure event (*bad*), then

$$Pr_{\mathbf{Game}_1}[P \wedge \neg bad] = Pr_{\mathbf{Game}_2}[P \wedge \neg bad]$$

Syntactic test implemented in Coq

- $Pr_{\mathbf{Game}_1}[\neg bad] = Pr_{\mathbf{Game}_2}[\neg bad]$
- $Pr_{\mathbf{Game}_1}[bad] = Pr_{\mathbf{Game}_2}[bad]$
  (if **Game**$_1$ and **Game**$_2$ are lossless)
- (Fundamental lemma):
  $\forall\ S\ \cdot\ |Pr_{\mathbf{Game}_1}[S] - Pr_{\mathbf{Game}_2}[S]| \leq Pr_{\mathbf{Game}_{1,2}}[bad]$

Coin fixing

Let $C[\cdot]$ be a context, c_1 and c_2 two sequences of instructions, e a boolean expression, and y a variable:

$$\{= \wedge e_{|1}\} \begin{array}{c} C[\text{if } e \text{ then } y \stackrel{\$}{\leftarrow} d; c_1 \text{ else } c_2] \\ (y \stackrel{\$}{\leftarrow} d; C[\text{if } e \text{ then } c_1 \text{ else } c_2]) \end{array} \{=_{Y \setminus \{y\}}\}$$

- c_2 does not reset e if it is false:

$$\{e = \mathit{false}\} c_2 \{e = \mathit{false}\}$$

- c_1 can "swap" with the random assignment of y:

$$\{= \wedge e_{|1}\} \ (c_1; \text{if } e \text{ then } y \stackrel{\$}{\leftarrow} d) \ (y \stackrel{\$}{\leftarrow} d; c_1) \ \{=\}$$

- C does not modify $fv(e, d)$ and does not read/write y

Essential tool to reason about random oracles

CertiCrypt: a user perspective

Trusting a machine-checked proof

CertiCrypt is designed to minimize the Trusted Computing Base.
To trust a proof in CertiCrypt, you must trust

- Libraries: probabilities, groups, polynomials
- Semantics of programs: execution, complexity, termination
- Statement of theorem: initial game, security properties,
- . . . Coq type checker

You need not trust or look at

- Tactics
- Proofs
- Intermediate games

By the way

- We do not have any user
- We are not likely to have users soon

Related work

- CryptoVerif
- Strongest postcondition for one-way function and random oracle
- Symbolic BPW model in Isabelle/HOL
- Formalisation of game-based proofs in Isabelle
- ElGamal and Switching Lemma in Coq
- Computational soundness in Coq

Conclusions

- CertiCrypt is a framework for machine checking game-based proofs in Coq
  - Core libraries are fully verified (25,000 lines of Coq)
  - Examples (OAEP, FDH, ElGamal)
    $\Rightarrow$ show the framework can be applied at reasonable cost?
- Much work remains to be done
  - More: case studies, automation
  - Soundness proofs: Dolev-Yao, inf. flow type systems, proof systems
  - Reasoning about randomized programs
- Exciting verification work. Will it impact cryptography?