Programming Language Techniques for Cryptographic Proofs

Gilles Barthe[1]
Benjamin Grégoire[2] Santiago Zanella-Béguelin[1]

[1]IMDEA Software, Madrid, Spain

[2]INRIA Sophia Antipolis - Méditerranée, France

ITP 2010

Formal verification of cryptographic primitives

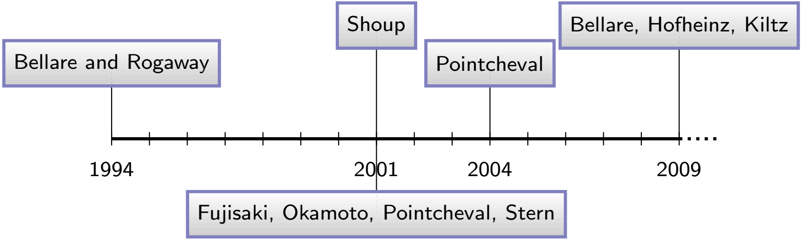
Security of cryptographic primitives is hard to achieve:

- "Secure schemes" broken after more than 10 years
- "Security proofs" remaining flawed over more than 15 years

First step: acknowledging the problem

- *Do we have a problem with cryptographic proofs? Yes, we do [...] We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect)*—Halevi, 2005
- *In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor*—Bellare and Rogaway, 2006

(In)Famous example: RSA-OAEP



1994 Purported proof of chosen-ciphertext security

2001 Proof is flawed, but can be patched

1. ...for a weaker security notion, or
2. ...for a modified scheme, or
3. ...under stronger assumptions

2004 Filled gaps in Fujisaki et al. 2001 proof

2009 Security definition needs to be clarified

2010 Filled gaps and marginally improved bound in 2004 proof

Exact IND-CCA security of OAEP

Game IND-CCA :
$(pk, sk) \leftarrow \mathcal{KG}(\eta);$
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
$b \stackrel{\$}{\leftarrow} \{0,1\};$
$c^* \leftarrow \mathcal{E}(m_b);$
$\tilde{b} \leftarrow \mathcal{A}_2(c^*)$

Game PD-OW :
$(pk, sk) \leftarrow \mathcal{KG}_f(\eta);$
$s \stackrel{\$}{\leftarrow} \{0,1\}^{n+k_1};$
$t \stackrel{\$}{\leftarrow} \{0,1\}^{k_0};$
$\tilde{s} \leftarrow \mathcal{I}(f(pk, s \parallel t))$

Security statement

$\forall \mathcal{A}, \exists \mathcal{I},$

$$2\left|\Pr[\text{IND-CPA} : \tilde{b} = b] - \frac{1}{2}\right| \leq$$
$$q_{\mathsf{H}}\Pr[\text{PD-OW} : \tilde{s} = s] + \frac{3q_{\mathcal{D}}q_{\mathsf{G}} + q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + q_{\mathsf{G}}}{2^{k_0}} + \frac{2q_{\mathcal{D}}}{2^{k_1}}$$

The proof has been machine-checked in the Coq proof assistant.

How?

Exact IND-CCA security of OAEP

Game IND-CCA :
$(pk, sk) \leftarrow \mathcal{KG}(\eta)$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^* \leftarrow \mathcal{E}(m_b)$;
$\tilde{b} \leftarrow \mathcal{A}_2(c^*)$

Game PD-OW :
$(pk, sk) \leftarrow \mathcal{KG}_f(\eta)$;
$s \xleftarrow{\$} \{0,1\}^{n+k_1}$;
$t \xleftarrow{\$} \{0,1\}^{k_0}$;
$\tilde{s} \leftarrow \mathcal{I}(f(pk, s \parallel t))$

Security statement

$\forall \mathcal{A}, \exists \mathcal{I}$,

$$2\left|\Pr[\text{IND-CPA} : \tilde{b} = b] - \frac{1}{2}\right| \leq$$
$$q_{\mathsf{H}} \Pr[\text{PD-OW} : \tilde{s} = s] + \frac{3q_{\mathcal{D}}q_{\mathsf{G}} + q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + q_{\mathsf{G}}}{2^{k_0}} + \frac{2q_{\mathcal{D}}}{2^{k_1}}$$

The proof has been machine-checked in the Coq proof assistant.

How?

Exact IND-CCA security of OAEP

Game IND-CCA :
$\mathbf{L_G}, \mathbf{L_H}, \mathbf{L_{\mathcal{D}}} \leftarrow d$;
$(\mathbf{pk}, sk) \leftarrow \mathcal{KG}(\eta)$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(\mathbf{pk})$;
$b \stackrel{\$}{\leftarrow} \{0,1\}$;
$c^* \leftarrow \mathcal{E}(m_b)$;
$\mathbf{c^*_{def}} \leftarrow$ true;
$\tilde{b} \leftarrow \mathcal{A}_2(c^*)$

Oracle $G(r)$:
if $r \notin \mathrm{dom}(\mathbf{L_G})$ then
$\mathbf{L_G}[r] \stackrel{\$}{\leftarrow} \{0,1\}^{n+k_1}$;
return $\mathbf{L_G}[r]$

Oracle $H(r)$: ...

Oracle $\mathcal{D}(c)$:
$\mathbf{L_{\mathcal{D}}} \leftarrow (\mathbf{c^*_{def}}, c) :: \mathbf{L_{\mathcal{D}}}$;
...

Security statement

$$\forall \mathcal{A}, \exists \mathcal{I},\ WF(\mathcal{A}) \wedge$$

$$\Pr\left[\text{IND-CCA} : \begin{array}{l} |\mathbf{L_G}| \leq q_{\mathsf{G}} + q_{\mathcal{D}} \wedge |\mathbf{L_H}| \leq q_{\mathsf{H}} \wedge |\mathbf{L_{\mathcal{D}}}| \leq q_{\mathcal{D}} \\ \wedge\ (\text{true}, c^*) \notin \mathbf{L_{\mathcal{D}}} \end{array}\right] = 1$$

$$\Longrightarrow\ 2\left|\Pr[\text{IND-CCA} : \tilde{b} = b] - \frac{1}{2}\right| \leq$$

$$q_{\mathsf{H}} \Pr[\text{PD-OW} : \tilde{s} = s] + \frac{3q_{\mathcal{D}}q_{\mathsf{G}} + q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + q_{\mathsf{G}}}{2^{k_0}} + \frac{2q_{\mathcal{D}}}{2^{k_1}}$$

The game-playing methodology

- How do we formalize the statement?

The game-playing methodology

- Games = (Families of) Probabilistic programs

Game G_0^η :
$\dots$
$\dots \leftarrow \mathcal{A}(\dots);$
$\dots$

$\Pr_{\mathsf{G}_0^\eta}[A_0]$

The game-playing methodology

- Games = (Families of) Probabilistic programs
- How do we perform the proof?

Game G_0^η :

$\ldots$

$\ldots \leftarrow \mathcal{A}(\ldots);$

$\ldots$

$\Pr_{\mathsf{G}_0^\eta}[A_0]$

The game-playing methodology

- Games = (Families of) Probabilistic programs
- Game transformation = Program transformation

Game G_0^η :
$\ldots$
$\ldots \leftarrow \mathcal{A}(\ldots);$
$\ldots$

Game G_1^η :
$\ldots$
$\ldots$
$\ldots$

$\cdots$

Game G_n^η :
$\ldots$
$\ldots \leftarrow \mathcal{B}(\ldots);$
$\ldots$

$$\Pr_{\mathsf{G}_0^\eta}[A_0] \quad \leq \quad h_1(\Pr_{\mathsf{G}_1^\eta}[A_1]) \quad \leq \ \ldots \ \leq \quad h_n(\Pr_{\mathsf{G}_n^\eta}[A_n])$$

CertiCrypt: machine-checking provable security

Certified framework for checking exact provable security proofs in the Coq proof assistant

- A combination of general methods from programming languages and of cryptographic-specific tools
- Game-based methodology, natural to cryptographers
- Focus on exact security bounds
- Several case studies:
  - Encryption schemes: ElGamal, Hashed ElGamal, OAEP, IBE
  - Signature schemes: FDH, BLS
  - Zero-knowledge proofs: see talk at CSF!

Inside CertiCrypt

- Semantics and cost model of probabilistic programs
- Model for adversaries
- Standard tools to reason about probabilistic programs
  - Semantics-preserving program transformations
  - Observational equivalence
  - Relational Hoare Logic
- In this talk: automation of 2 reasoning patterns in crypto:
  1. Bounding failure events
  2. Moving sampling of random values accross procedures

pWhile: a probabilistic programming language

$$
\begin{array}{rcll}
\mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
& | & \mathcal{V} \xleftarrow{\$} \mathcal{DE} & \text{random sampling} \\
& | & \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
& | & \text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
& | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} & ::= & \text{skip} & \text{nop} \\
& | & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}
$$

$x \xleftarrow{\$} d$: sample x according to distribution d, typically the uniform distribution on a set (e.g. $\{0,1\}$, $\{0,1\}^{\ell}$)

Deep Embedding

The syntax of programs is formalized as an inductive type

Dependently-typed Syntax

$$
\begin{array}{l}
\textbf{Inductive}\ \mathcal{I} := \\
\mid \text{Assign} : \forall t,\ \mathcal{V}_t \rightarrow \mathcal{E}_t \rightarrow \mathcal{I} \\
\mid \text{Rand} \quad : \forall t,\ \mathcal{V}_t \rightarrow \mathcal{DE}_t \rightarrow \mathcal{I} \\
\mid \text{Cond} \quad : \mathcal{E}_{\mathbb{B}} \rightarrow \mathcal{C} \rightarrow \mathcal{C} \rightarrow \mathcal{I} \\
\mid \text{While} \quad : \mathcal{E}_{\mathbb{B}} \rightarrow \mathcal{C} \rightarrow \mathcal{I} \\
\mid \text{Call} \quad : \forall l\ t, \mathcal{P}_{(l,t)} \rightarrow\ \mathcal{V}_t \rightarrow \text{dlist}\ l\ \mathcal{E} \rightarrow \mathcal{I} \\
\textbf{where}\ \mathcal{C} := \text{list}\ \mathcal{I}
\end{array}
$$

- Programs are well-typed by construction
- Semantics as a total function
- Allows richer specification (e.g. enforce size constraints on bitstrings)

Semantics

Measure Monad —courtesy of Christine Paulin

Distributions represented as functions of type

$$\mathcal{D}(A) \stackrel{\text{def}}{=} (A \to [0,1]) \to [0,1] \quad \text{s.t.}$$

1. $f \leq g \implies \mu(f) \leq \mu(g)$;
2. $\mu(\mathbb{1} - f) \leq 1 - \mu(f)$;
3. $f \leq \mathbb{1} - g \implies \mu(f+g) = \mu(f) + \mu(g)$;
4. $\mu(k \times f) = k \times \mu(f)$;
5. $f : \mathbb{N} \to (A \to [0,1])$ is monotonic and for all $n \in \mathbb{N}$ $f(n)$ is monotonic, then $\mu(\sup\ f) \leq \sup\ (\lambda n.\ \mu(f(n)))$

All arithmetic is in the unit interval $[0,1]$

$$\begin{array}{llcl} \text{unit} & : A \to \mathcal{D}(A) & \stackrel{\text{def}}{=} & \lambda x.\ \lambda f.\ f\ x \\ \text{bind} & : \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B) & \stackrel{\text{def}}{=} & \lambda \mu.\ \lambda F.\ \lambda f.\ \mu(\lambda x.\ F\ x\ f) \end{array}$$

Semantics

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

$$
\begin{array}{lcl}
[\![\mathsf{skip}]\!] & = & \mathsf{unit} \\
[\![i;\ c]\!]\ m & = & \mathsf{bind}\ ([\![i]\!]\ m)\ [\![c]\!] \\
[\![x \leftarrow e]\!]\ m & = & \mathsf{unit}\ m\{([\![e]\!]_{\mathcal{E}}\ m)/x\} \\
[\![x \xleftarrow{\$} d]\!]\ m & = & \mathsf{bind}\ ([\![d]\!]_{\mathcal{DE}}\ m)\ (\lambda v.\ \mathsf{unit}\ m\{v/x\}) \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\ m & = & \begin{cases} [\![c_1]\!]\ m & \text{if } [\![e]\!]_{\mathcal{E}}\ m = \mathsf{true} \\ [\![c_2]\!]\ m & \text{if } [\![e]\!]_{\mathcal{E}}\ m = \mathsf{false} \end{cases} \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\ m & = & \lambda f.\mathsf{sup}\ (\lambda n.\ [\![[\mathsf{while}\ e\ \mathsf{do}\ c]_n]\!]\ m\ f)
\end{array}
$$

where

$$
\begin{array}{lcl}
[\mathsf{while}\ e\ \mathsf{do}\ c]_0 & = & \mathsf{skip} \\
[\mathsf{while}\ e\ \mathsf{do}\ c]_{n+1} & = & \mathsf{if}\ e\ \mathsf{then}\ c;\ [\mathsf{while}\ e\ \mathsf{do}\ c]_n
\end{array}
$$

$$[\![x \leftarrow p(\vec{e})]\!]\ m \quad = \quad \mathsf{bind}\ ([\![p.\mathsf{body}]\!]\ \ldots$$

Not axioms: actual function built from small-step semantics

Semantics

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \rightarrow \mathcal{D}(\mathcal{M})$$

$$
\begin{array}{lcl}
[\![\mathsf{skip}]\!] & = & \mathsf{unit} \\
[\![i;\ c]\!]\ m & = & \mathsf{bind}\ ([\![i]\!]\ m)\ [\![c]\!] \\
[\![x \leftarrow e]\!]\ m & = & \mathsf{unit}\ m\{([\![e]\!]_{\mathcal{E}}\ m)/x\} \\
[\![x \xleftarrow{\$} d]\!]\ m & = & \mathsf{bind}\ ([\![d]\!]_{\mathcal{DE}}\ m)\ (\lambda v.\ \mathsf{unit}\ m\{v/x\}) \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\ m & = & \begin{cases} [\![c_1]\!]\ m & \text{if } [\![e]\!]_{\mathcal{E}}\ m = \mathsf{true} \\ [\![c_2]\!]\ m & \text{if } [\![e]\!]_{\mathcal{E}}\ m = \mathsf{false} \end{cases} \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\ m & = & \lambda f.\, \mathsf{sup}\ (\lambda n.\ [\![[\mathsf{while}\ e\ \mathsf{do}\ c]_n]\!]\ m\ f)
\end{array}
$$

where

$$
\begin{array}{lcl}
[\mathsf{while}\ e\ \mathsf{do}\ c]_0 & = & \mathsf{skip} \\
[\mathsf{while}\ e\ \mathsf{do}\ c]_{n+1} & = & \mathsf{if}\ e\ \mathsf{then}\ c;\ [\mathsf{while}\ e\ \mathsf{do}\ c]_n
\end{array}
$$

$$[\![x \leftarrow p(\vec{e})]\!]\ m \quad = \quad \mathsf{bind}\ ([\![p.\mathsf{body}]\!]\ \ldots$$

Not axioms: actual function built from small-step semantics

Observational Equivalence

Games G_1 and G_2 are observationally equivalent w.r.t. input variables I and output variables O iff:

- IF m_1 and m_2 coincide on I
- THEN $[\![\mathsf{G}_1]\!]\ m_1$ and $[\![\mathsf{G}_2]\!]\ m_2$ coincide on O (i.e. their projections on O are equal)

$$
\begin{aligned}
m_1 =_X m_2 &\stackrel{\text{def}}{=} \forall x \in X,\ m_1\ x = m_2\ x \\
f =_X g &\stackrel{\text{def}}{=} \forall m_1\ m_2,\ m_1 =_X m_2 \implies f\ m_1 = g\ m_2 \\
\models \mathsf{G}_1 \simeq^I_O \mathsf{G}_2 &\stackrel{\text{def}}{=} \forall m_1\ m_2\ f\ g,\ m_1 =_I m_2\ \wedge\ f =_O g \implies \\
& \qquad [\![\mathsf{G}_1]\!]\ m_1\ f = [\![\mathsf{G}_2]\!]\ m_2\ g
\end{aligned}
$$

- Generalized to arbitrary relations
- Probabilistic Relational Hoare Logic

...but this is not what this talk is about

Reasoning about Failure Events

Lemma (Fundamental Lemma of Game-Playing)

Let A, B, F *be events and* $\mathsf{G}_1, \mathsf{G}_2$ *be two games such that*

$$\Pr[\mathsf{G}_1 : A \wedge \neg F] = \Pr[\mathsf{G}_2 : B \wedge \neg F]$$

Then, $|\Pr[\mathsf{G}_1 : A] - \Pr[\mathsf{G}_2 : B]| \leq \max(\Pr[\mathsf{G}_1 : F], \Pr[\mathsf{G}_2 : F])$

Automation

Syntectic Criterion

When $A = B$ and $F =$ bad. If G_0, G_1 are syntactically identical except after program points setting bad e.g.

Game G_0 :
...
bad $\leftarrow$ true; c_0
...

Game G_1 :
...
bad $\leftarrow$ true; c_1
...

...and bad is never reset, then

- $\Pr[G_0 : A \wedge \neg\text{bad}] = \Pr[G_1 : A \wedge \neg\text{bad}]$
- If game G_i (c_i) terminates with probability 1:
  $\Pr[G_{1-i} : \text{bad}] \leq \Pr[G_i : \text{bad}]$
- If both c_0, c_1 terminate absolutely:
  $\Pr[G_0 : \text{bad}] = \Pr[G_1 : \text{bad}]$

Automation

Syntectic Criterion

When $A = B$ and $F = \text{bad}$. If G_0, G_1 are syntactically identical except after program points setting bad e.g.

Game G_0 :

$\ldots$

bad $\leftarrow$ true; c_0

$\ldots$

Game G_1 :

$\ldots$

bad $\leftarrow$ true; c_1

$\ldots$

...and bad is never reset, then

- $\Pr[G_0 : A \wedge \neg\text{bad}] = \Pr[G_1 : A \wedge \neg\text{bad}]$
- If game G_i (c_i) terminates with probability 1: $\Pr[G_{1-i} : \text{bad}] \leq \Pr[G_i : \text{bad}]$
- If both c_0, c_1 terminate absolutely: $\Pr[G_0 : \text{bad}] = \Pr[G_1 : \text{bad}]$

Failure Event lemma

Motivation: the Fundamental Lemma is typically applied in games where only oracles trigger bad.

- IF the probability of triggering bad in an oracle call can be bound as a function of the number of oracle calls so far
- THEN the probability of the whole game triggering bad can be bound if the number of oracle calls is bounded

Failure Event Lemma (constant case)

Assume that $m(\text{bad}) = \text{false}$

- IF $\Pr[\mathcal{O}, m : \text{bad}] \leq p$ for every memory m such that $m(\text{bad}) = \text{false}$
- THEN $\Pr[\mathsf{G}, m : \text{bad}] \leq p\ q_{\mathcal{O}}$

Hypothesis holds for oracle

$$\mathcal{O}(x) : y \xleftarrow{\$} T; \text{if } y = y_0 \text{ then bad} \leftarrow \text{true else } \ldots$$

with $p = 1/|T|$

Logic of Failure Events

A variant of Probabilistic Hoare Logic

$$\vdash [\![c]\!]g \preceq f \quad \stackrel{\text{def}}{=} \quad \forall m. [\![c]\!]\ m\ g \leq f(m)$$

Selected Rules

$$\vdash [\![\text{skip}]\!]f \preceq f \qquad \vdash [\![x \leftarrow e]\!]g \preceq \lambda m.\ g(m\{[\![e]\!]\ m/x\})$$

$$\vdash [\![x \overset{\$}{\leftarrow} T]\!]g \preceq \lambda m. |[\![T]\!]|^{-1} \textstyle\sum_{t \in [\![T]\!]}\ g(m\{t/x\})$$

$$\frac{\vdash [\![c_1]\!]g \preceq f \quad [\![c_2]\!]h \preceq g}{\vdash [\![c_1; c_2]\!]h \preceq f} \qquad \frac{\vdash [\![c_1]\!]g \preceq f \quad [\![c_2]\!]g \preceq f}{\vdash [\![\text{if } e \text{ then } c_1 \text{ else } c_2]\!]g \preceq f}$$

$$\frac{f = \qquad I \vDash c \simeq^I_O c' \quad g = \qquad O \vdash [\![c']\!]g \preceq f}{\vdash [\![c]\!]g \preceq f}$$

Relation to Hoare Logic (for Boolean-valued P, Q):

Partial correctness: $\{P\}c\{Q\} \iff [\![c]\!]\mathbb{1}_{\neg Q} \preceq \mathbb{1}_{\neg P}$

Total correctness: $\{P\}c\{Q\} \iff [\![c]\!]\mathbb{1}_{Q} \succeq \mathbb{1}_{P}$

Logic of Failure Events

A variant of Probabilistic Hoare Logic

$$\vdash [\![c]\!]g \preceq f \quad \stackrel{\text{def}}{=} \quad \forall m. [\![c]\!]\ m\ g \leq f(m)$$
$$\vdash [\![c]\!]g \succeq f \quad \stackrel{\text{def}}{=} \quad \forall m. [\![c]\!]\ m\ g \geq f(m)$$

Selected Rules

$$\vdash [\![\text{skip}]\!]f \preceq f \qquad \vdash [\![x \leftarrow e]\!]g \preceq \lambda m.\ g(m\{[\![e]\!]\ m/x\})$$

$$\vdash [\![x \stackrel{\$}{\leftarrow} T]\!]g \preceq \lambda m. |[\![T]\!]|^{-1} \textstyle\sum_{t \in [\![T]\!]}\ g(m\{t/x\})$$

$$\frac{\vdash [\![c_1]\!]g \preceq f \quad [\![c_2]\!]h \preceq g}{\vdash [\![c_1; c_2]\!]h \preceq f} \qquad \frac{\vdash [\![c_1]\!]g \preceq f \quad [\![c_2]\!]g \preceq f}{\vdash [\![\text{if } e \text{ then } c_1 \text{ else } c_2]\!]g \preceq f}$$

$$\frac{f = \quad I \vDash c \simeq^I_O c' \quad g = \quad O \vdash [\![c']\!]g \preceq f}{\vdash [\![c]\!]g \preceq f}$$

Relation to Hoare Logic (for Boolean-valued P, Q):

Partial correctness:	$\{P\}c\{Q\}$	$\iff$	$[\![c]\!]\mathbb{1}_{\neg Q} \preceq \mathbb{1}_{\neg P}$
Total correctness:	$\{P\}c\{Q\}$	$\iff$	$[\![c]\!]\mathbb{1}_{Q} \succeq \mathbb{1}_{P}$

Application: PRP/PRF Switching Lemma

Game G_{RP} :
$\mathbf{L} \leftarrow \mathsf{nil};\ b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:
if $x \notin \mathsf{dom}(\mathbf{L})$ then
  $y \xleftarrow{\$} \{0,1\}^{\ell} \setminus \mathsf{ran}(\mathbf{L})$;
  $\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$
return $\mathbf{L}(x)$

Game G_{RF} :
$\mathbf{L} \leftarrow \mathsf{nil};\ b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:
if $x \notin \mathsf{dom}(\mathbf{L})$ then
  $y \xleftarrow{\$} \{0,1\}^{\ell}$;
  $\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$
return $\mathbf{L}(x)$

Suppose $\mathcal{A}$ makes at most q queries to $\mathcal{O}$. Then

$$|\Pr[\mathsf{G}_{\mathsf{RP}} : b] - \Pr[\mathsf{G}_{\mathsf{RF}} : b]| \leq \frac{q(q-1)}{2^{\ell+1}}$$

- First introduced by Impagliazzo and Rudich in 1989
- Proof fixed by Bellare and Rogaway (2006) and Shoup (2004)

Proof

Game G_{RP} :
$\mathbf{L} \leftarrow \mathsf{nil};\ b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:
if $x \notin \mathsf{dom}(\mathbf{L})$ then
  $y \stackrel{\$}{\leftarrow} \{0,1\}^\ell$;
  if $y \in \mathsf{ran}(\mathbf{L})$ then ;
    $\mathsf{bad} \leftarrow \mathsf{true}$;
    $y \stackrel{\$}{\leftarrow} \{0,1\}^\ell \setminus \mathsf{ran}(\mathbf{L})$
  $\mathbf{L} \leftarrow (x,y) :: \mathbf{L}$
return $\mathbf{L}(x)$

Game G_{RF} :
$\mathbf{L} \leftarrow \mathsf{nil};\ b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:
if $x \notin \mathsf{dom}(\mathbf{L})$ then
  $y \stackrel{\$}{\leftarrow} \{0,1\}^\ell$;
  if $y \in \mathsf{ran}(\mathbf{L})$ then ;
    $\mathsf{bad} \leftarrow \mathsf{true}$
  $\mathbf{L} \leftarrow (x,y) :: \mathbf{L}$
return $\mathbf{L}(x)$

$$|\Pr[\mathsf{G}_{\mathsf{RP}} : b] - \Pr[\mathsf{G}_{\mathsf{RF}} : b]| \leq \Pr[\mathsf{G}_{\mathsf{RF}} : \mathsf{bad}]$$

Proof

Failure Event Lemma (less simplified)

Let k be a counter for $\mathcal{O}$ and $m(\text{bad}) = \text{false}$:

- IF $\Pr[\mathcal{O}, m : \text{bad}] \leq f(m(k))$ for all memories m such that $m(\text{bad}) = \text{false}$
- THEN $\Pr[\mathsf{G}, m : \text{bad}] \leq \sum_{k=0}^{q_{\mathcal{O}}-1} f(k)$

Oracle $\mathcal{O}(x)$:
if $x \notin \text{dom}(\mathbf{L})$ then
  $y \xleftarrow{\$} \{0,1\}^{\ell}$; if $y \in \text{ran}(\mathbf{L})$ then bad $\leftarrow$ true;
  $\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$
return $\mathbf{L}(x)$

- Prove that

$$\Pr[\mathsf{G}, m : \text{bad}] \leq \frac{|m(\mathbf{L})|}{2^{\ell}}$$

Eager/Lazy Sampling

- Interprocedural code motion
- Eager sampling: from an oracle to main game
- Lazy sampling: from main game to an oracle

Motivation

In crypto proofs

- Often need to know that some values are independent and uniformly distributed at some program point
- This holds when values can be resampled preserving semantics!

To prove correctness of eager and lazy sampling, we developed a logic for swapping statements

$$\models E,\ (c; S) \simeq E',\ (S; c')$$

Selected Rules

Assume $\mathsf{modifies}(E, S) \cup \mathsf{modifies}(E', S) \subseteq X$ and $\models E, S \simeq_X^X E', S$

$$\frac{x \notin X \qquad \mathsf{fv}(e) \cap X = \emptyset}{\models E, (x \leftarrow e; S) \equiv E', (S; x \leftarrow e)}$$

$$\frac{x \notin X}{\models E, (x \xleftarrow{\$} T; S) \equiv E', (S; x \xleftarrow{\$} T)}$$

$$\frac{\models E, (c_1; S) \equiv E', (S; c_1') \quad \models E, (c_2; S) \equiv E', (S; c_2')}{\models E, (c_1; c_2; S) \equiv E', (S; c_1'; c_2')}$$

$$\frac{\models E, (c_1; S) \equiv E', (S; c_1') \qquad \models E, (c_2; S) \equiv E', (S; c_2') \qquad \mathsf{fv}(e) \cup X = \emptyset}{\models E, (\text{if } e \text{ then } c_1 \text{ else } c_2; S) \equiv E', (S; \text{if } e \text{ then } c_1' \text{ else } c_2')}$$

Application: PRP/PRF Switching Lemma

Game $\mathsf{G}_{\mathsf{RF}}^{\mathrm{eager}}$:
$\mathbf{L} \leftarrow \mathsf{nil};\ S;\ b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:
if $x \notin \mathsf{dom}(\mathbf{L})$ then
  if $0 < |\mathbf{Y}|$ then
    $y \leftarrow \mathsf{hd}(\mathbf{Y});\ \mathbf{Y} \leftarrow \mathsf{tl}(\mathbf{Y})$
  else $y \xleftarrow{\$} \{0,1\}^\ell$
  $\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$
return $\mathbf{L}(x)$

where $S \stackrel{\text{def}}{=} \mathbf{Y} \leftarrow [\,];$ while $|\mathbf{Y}| < q$ do $y \xleftarrow{\$} \{0,1\}^\ell;\ \mathbf{Y} \leftarrow \mathbf{Y} +\!\!+ [y]$

Prove using the logic:

$$\vDash E_{RF},\ (b \leftarrow \mathcal{A}();\ S) \equiv E_{RF}^{\mathrm{eager}},\ (S;\ b \leftarrow \mathcal{A}())$$

Prove by induction:

$$\Pr[\mathsf{G}_{\mathsf{RF}}; S : \mathsf{bad}] = \Pr[\mathsf{G}_{\mathsf{RF}}^{\mathrm{eager}} : \mathsf{collision}] = \sum_{i=0}^{q-1} \frac{i}{2^\ell}$$

Summary

CertiCrypt: crypto proofs using programming language techniques

- Observational equivalence
- Relational Hoare Logic
- Certified program transformations

...including a few non-standard techniques

- Failure events
- Eager and lazy sampling

Tools in this paper increase automation and abstraction.
Proof of the PRP/PRF Switching Lemma:

- Original (POPL'09): 900 lines
- Using logic of swapping statements: 400 lines
- Using Failure Event Lemma: 100 lines

The road ahead

Increasing abstraction and automation will hopefully make verifiable security a reasonable and profitable alternative for cryptographers (see FCC'10 talk next week)