EasyCrypt – An Interactive Proof Assistant with Support for Cryptographic Proofs

François Dupressoir (U. of Surrey)
with G. Barthe (IMDEA Software Institute), B. Grégoire (Inria Sophia Antipolis), P.-Y. Strub (École Polytechnique), B. Schmidt (Google) and others

12-13 July 2017
Summer School on Models and Tools for Cryptographic Proofs, Nancy

Problems with cryptographic proofs

Unverifiable proofs

- Proofs are long and error-prone
- Rely on unstated and unverified invariants
- Intricate reasoning steps justified informally

[...] many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor.
Bellare and Rogaway, 2004-2006

Abstraction gap(s)

- Provable security reasons about algorithmic descriptions
- Standards constrain implementations
- Attackers target executable code

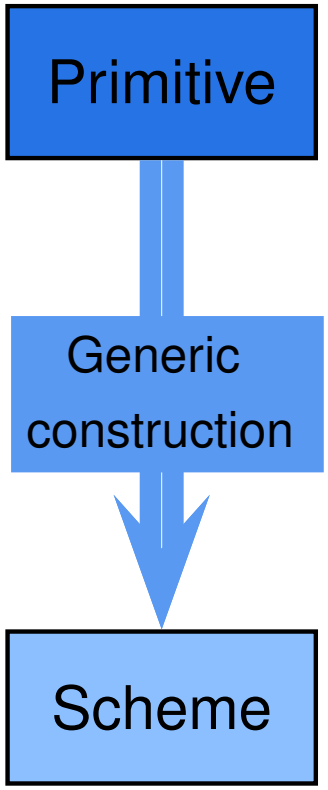
Real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security. Bernstein, 2013

Computer-aided cryptographic proofs

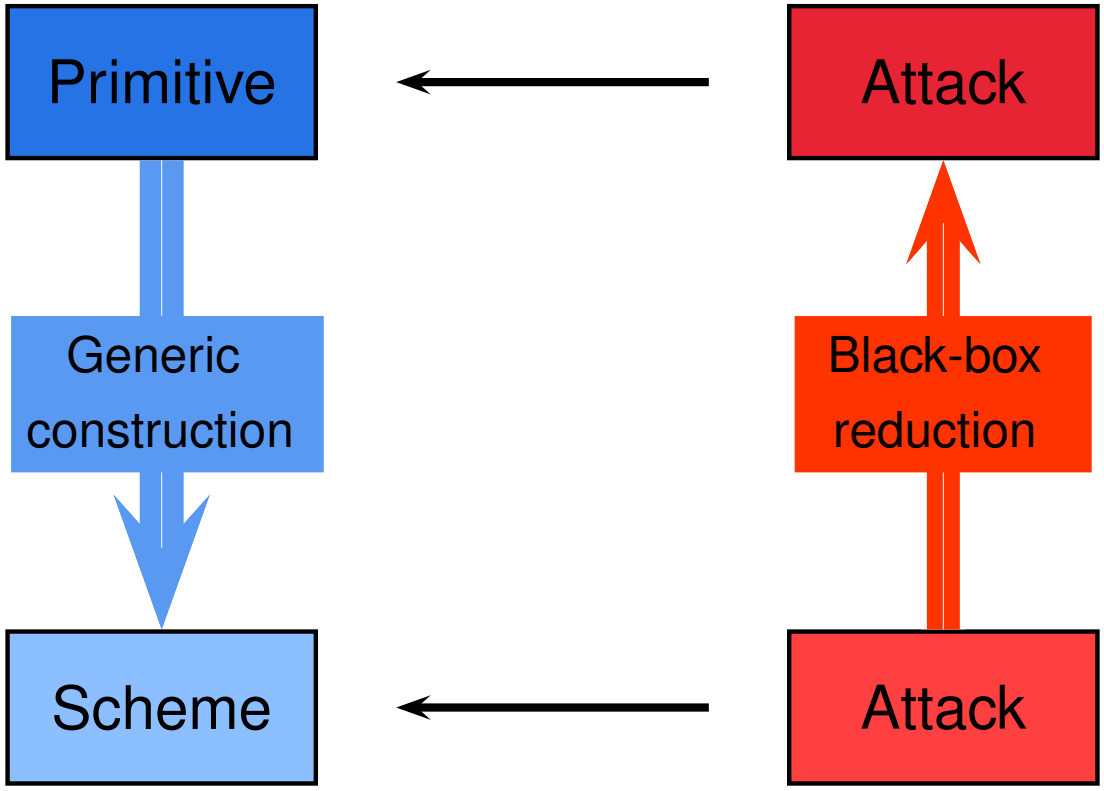
Provable security as deductive relational verification of open probabilistic parameterized programs

- High-confidence reduction- and simulation-based proofs
  - ☞ machine-checked, independently verifiable proofs
  - ☞ adhere to cryptographic practice (same formalisms, guarantees and proof techniques)
- Manage complexity of real-world cryptography
- Increase confidence in implementations
  - ☞ minimize gap between proofs and code
  - ☞ prove effectiveness of countermeasures (on source code and machine code)
- Leverage existing verification techniques and tools
  - ☞ program logics, VC generation, invariant generation
  - ☞ SMT solvers, theorem provers, proof assistants
  - ☞ verified compilers

Reductionist proof



Reductionist proof



An Example: Bellare and Rogaway 93 Public-Key Encryption

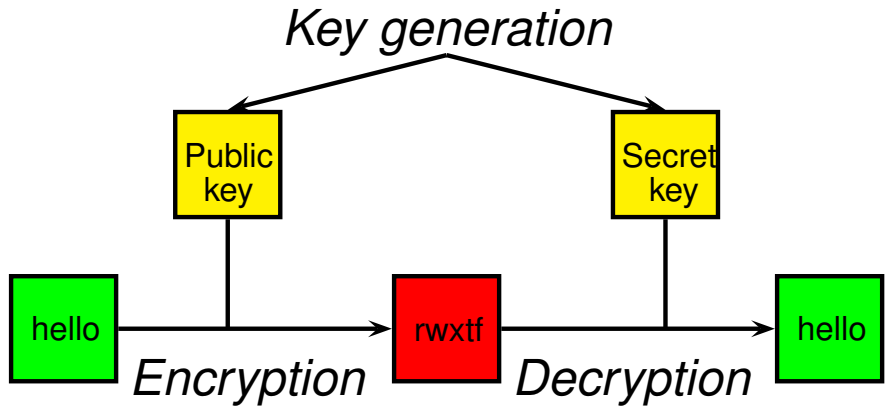
Public-key encryption

Algorithms $(\mathcal{K}, \mathcal{E}_{pk}, \mathcal{D}_{sk})$

- $\mathcal{E}$ probabilistic
- $\mathcal{D}$ deterministic and partial

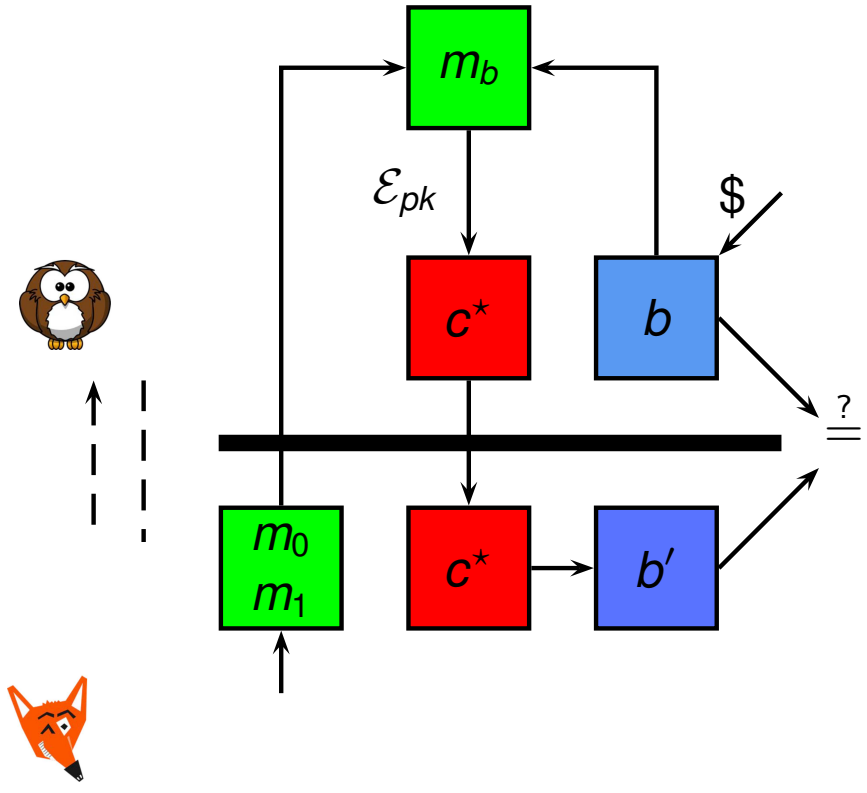
If (sk, pk) is a valid key pair,

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$$



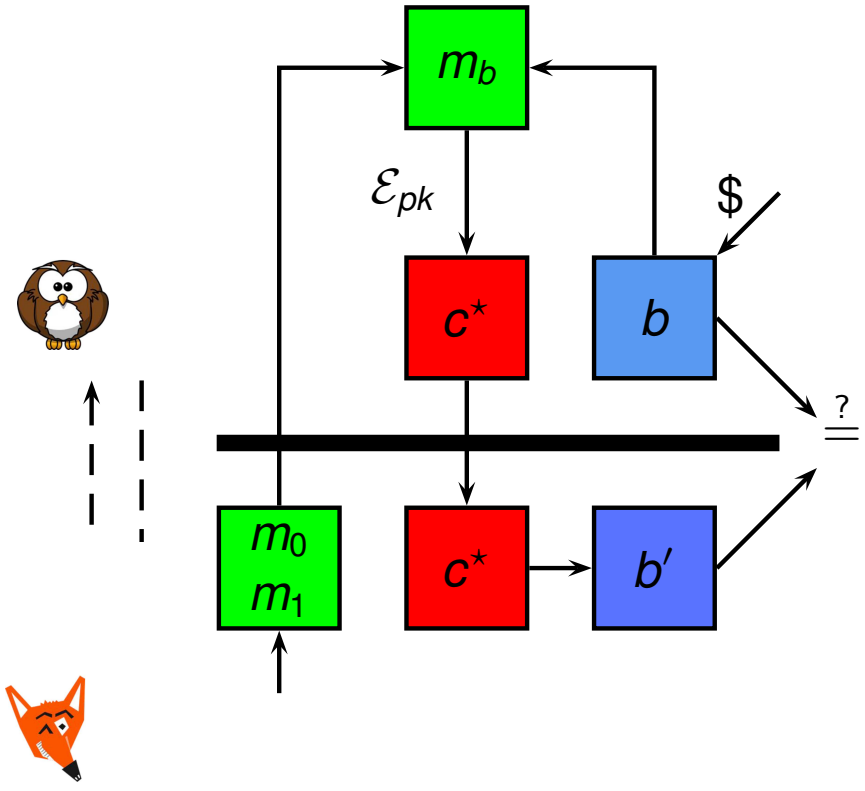
Indistinguishability

Game IND-CPA($\mathcal{A}$)
$(sk, pk) \leftarrow \mathcal{K}()$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
return $(b' = b)$



Indistinguishability

Game IND-CPA($\mathcal{A}$)
$(sk, pk) \leftarrow \mathcal{K}()$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
return $(b' = b)$



$$\Pr_{\text{IND-CPA}(\mathcal{A})}\left[b' = b\right] - \frac{1}{2} \quad \text{small}$$

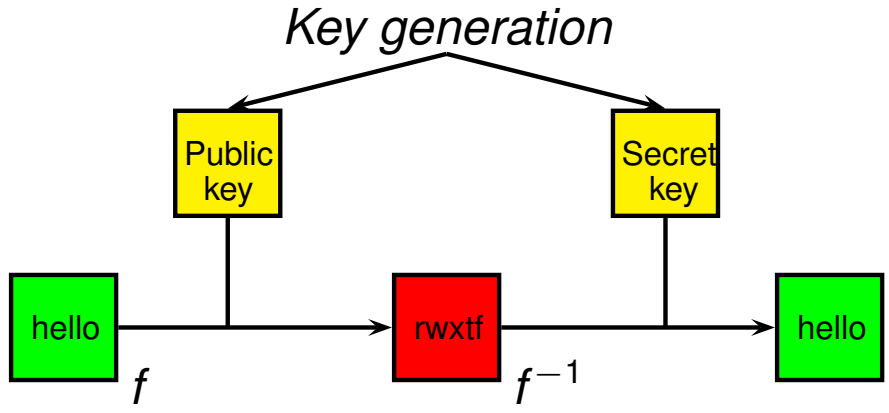
One-way trapdoor permutations

Algorithms $(\mathcal{K}, \mathsf{f}_{pk}, \mathsf{f}_{sk}^{-1})$

- f_{pk} and f_{sk}^{-1} deterministic

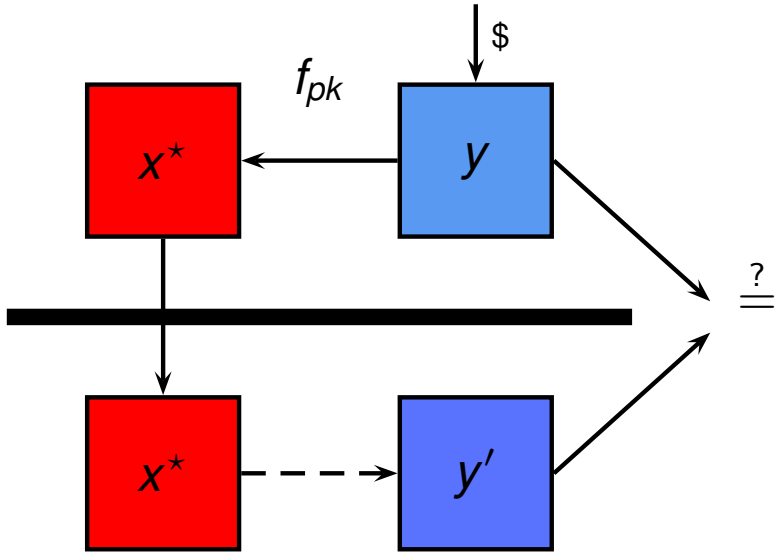
If (sk, pk) is a valid key pair,

$$\mathsf{f}_{sk}^{-1}(\mathsf{f}_{pk}(m)) = m$$



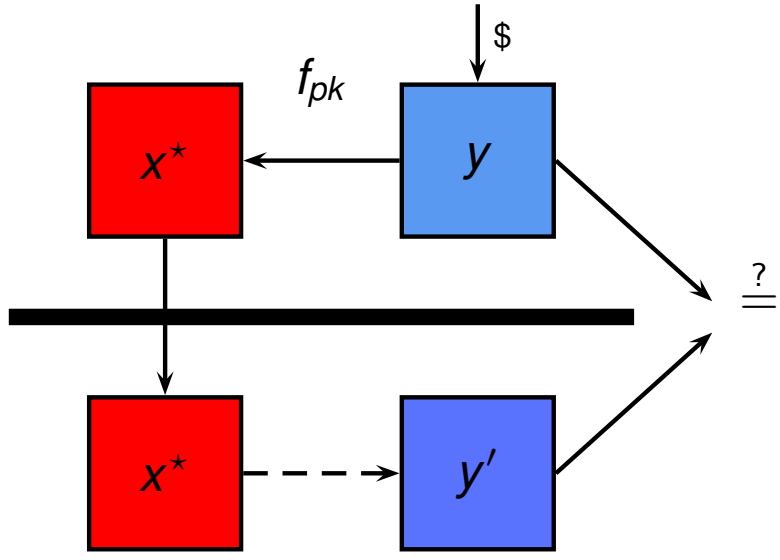
One-way trapdoor permutations

Game $\mathrm{OW}(\mathcal{I})$
$(sk, pk) \leftarrow \mathcal{K}();$
$y \xleftarrow{\$} \{0,1\}^n;$
$x^\star \leftarrow \mathsf{f}_{pk}(y);$
$y' \leftarrow \mathcal{I}(x^\star);$
return $(y' = y)$



One-way trapdoor permutations

Game $\mathrm{OW}(\mathcal{I})$
$(sk, pk) \leftarrow \mathcal{K}();$
$y \xleftarrow{\$} \{0,1\}^n;$
$x^\star \leftarrow \mathsf{f}_{pk}(y);$
$y' \leftarrow \mathcal{I}(x^\star);$
return $(y' = y)$



$$\Pr_{\mathrm{OW}(\mathcal{I})}\left[y' = y\right] \quad \text{small}$$

Random oracles

Oracle $H(x)$:
if $x \notin L$ then
  $r \xleftarrow{\$} \{0,1\}^k$;
  $L \leftarrow (x, r) :: L$;
return $L[x]$;

- Idealized model of hash function
- Allows practical schemes
- Not realizable

Example: Bellare and Rogaway 1993 encryption

Game IND-CPA($\mathcal{A}$) :
$(sk, pk) \leftarrow \mathcal{K}(\ )$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
return $(b' = b)$

Encryption $\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^\ell$;
$s \leftarrow H(r) \oplus m$;
$y \leftarrow f_{pk}(r) \,\|\, s$;
return y

For every IND-CPA adversary $\mathcal{A}$, there exists an inverter $\mathcal{I}$ st

$$\Pr_{\text{IND-CPA}(\mathcal{A})}\left[b' = b\right] - \frac{1}{2} \leq \Pr_{\text{OW}(\mathcal{I})}\left[y' = y\right]$$

Proof

Game hopping technique

Game INDCPA :
$(sk, pk) \leftarrow \mathcal{K}();$
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
$b \xleftarrow{\$} \{0,1\};$
$c^\star \leftarrow \mathcal{E}_{pk}(m_b);$
$b' \leftarrow \mathcal{A}_2(c^\star);$
return $(b' = b)$
Encryption $\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^\ell;$
$h \leftarrow H(r);$
$s \leftarrow h \oplus m;$
$c \leftarrow f_{pk}(r) \,\|\, s;$
return c

Game G :
$(sk, pk) \leftarrow \mathcal{K}();$
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
$b \xleftarrow{\$} \{0,1\};$
$c^\star \leftarrow \mathcal{E}_{pk}(m_b);$
$b' \leftarrow \mathcal{A}_2(c^\star);$
return $(b' = b)$
Encryption $\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^\ell;$
$h \xleftarrow{\$} \{0,1\}^k;$
$s \leftarrow h \oplus m;$
$c \leftarrow f_{pk}(r) \,\|\, s;$
return c

Game G′ :
$(sk, pk) \leftarrow \mathcal{K}();$
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
$b \xleftarrow{\$} \{0,1\};$
$c^\star \leftarrow \mathcal{E}_{pk}(m_b);$
$b' \leftarrow \mathcal{A}_2(c^\star);$
return $(b' = b)$
Encryption $\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^\ell;$
$s \xleftarrow{\$} \{0,1\}^k;$
$h \leftarrow s \oplus m;$
$c \leftarrow f_{pk}(r) \,\|\, s;$
return c

Game OW :
$(sk, pk) \leftarrow \mathcal{K}();$
$y \xleftarrow{\$} \{0,1\}^\ell;$
$y' \leftarrow \mathcal{I}(f_{pk}(y));$
return $y = y'$
Adversary $\mathcal{I}(x)$:
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
$s \xleftarrow{\$} \{0,1\}^k;$
$c^\star \leftarrow x \,\|\, s;$
$b' \leftarrow \mathcal{A}_2(c^\star);$
$y' \leftarrow [z \in L_H^{\mathcal{A}} \,|\, f_{pk}(z) = x];$
return y'

1. Prove a probability claim for each hop
2. Obtain security bound by combining claims
3. Check execution time of constructed adversary

Conditional equivalence

$\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{\ell}$;
$h \leftarrow H(r)$;
$s \leftarrow h \oplus m$;
$c \leftarrow f_{pk}(r) \,\|\, s$;
return c

$\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{\ell}$;
$h \xleftarrow{\$} \{0,1\}^{k}$;
$s \leftarrow h \oplus m$;
$c \leftarrow f_{pk}(r) \,\|\, s$;
return c

By the Fundamental Lemma

$$\Pr_{\text{IND-CPA}}\left[b' = b\right] - \Pr_{\mathbf{G}}\left[b' = b\right] \leq \Pr_{\mathbf{G}}\left[r \in L_H^{\mathcal{A}}\right]$$

Equivalence

$\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{\ell}$;
$h \xleftarrow{\$} \{0,1\}^{k}$;
$s \leftarrow h \oplus m$;
$c \leftarrow f_{pk}(r) \| s$;
return c

$\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{\ell}$;
$s \xleftarrow{\$} \{0,1\}^{k}$;
$h \leftarrow s \oplus m$;
$c \leftarrow f_{pk}(r) \| s$;
return c

By optimistic sampling

$$\Pr_{\mathbf{G}}\left[r \in L_H^{\mathcal{A}}\right] = \Pr_{\mathbf{G}'}\left[r \in L_H^{\mathcal{A}}\right] \qquad \Pr_{\mathbf{G}}[b' = b] = \Pr_{\mathbf{G}'}[b' = b] = \frac{1}{2}$$

Equivalence

$\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{\ell}$;
$h \xleftarrow{\$} \{0,1\}^{k}$;
$s \leftarrow h \oplus m$;
$c \leftarrow f_{pk}(r) \| s$;
return c

$\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{\ell}$;
$s \xleftarrow{\$} \{0,1\}^{k}$;
$h \leftarrow s \oplus m$;
$c \leftarrow f_{pk}(r) \| s$;
return c

By optimistic sampling

$$\Pr_{\text{IND-CPA}}[b' = b] - \tfrac{1}{2} \leq \Pr_{\mathbf{G}'}\left[r \in L_H^{\mathcal{A}}\right]$$

Reduction

Game INDCPA :
$(sk, pk) \leftarrow \mathcal{K}()$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
return $(b' = b)$
Encryption $\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^\ell$;
$s \xleftarrow{\$} \{0,1\}^k$;
$c \leftarrow f_{pk}(r) \,\|\, s$;
return c

Game OW :
$(sk, pk) \leftarrow \mathcal{K}()$;
$y \xleftarrow{\$} \{0,1\}^\ell$;
$y' \leftarrow \mathcal{I}(f_{pk}(y))$;
return $y = y'$
Adversary $\mathcal{I}(x)$:
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$s \xleftarrow{\$} \{0,1\}^k$;
$c^\star \leftarrow x \,\|\, s$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
$y' \leftarrow [z \in L_H^{\mathcal{A}} \mid f_{pk}(z) = x]$;
return y'

$$\Pr_{\mathbf{G}'}\left[r \in L_H^{\mathcal{A}}\right] \leq \Pr_{\mathrm{OW}(\mathcal{I})}[y' = y]$$

Reduction

Game INDCPA :
$(sk, pk) \leftarrow \mathcal{K}()$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
return $(b' = b)$
Encryption $\mathcal{E}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^\ell$;
$s \xleftarrow{\$} \{0,1\}^k$;
$c \leftarrow f_{pk}(r) \,\|\, s$;
return c

Game OW :
$(sk, pk) \leftarrow \mathcal{K}()$;
$y \xleftarrow{\$} \{0,1\}^\ell$;
$y' \leftarrow \mathcal{I}(f_{pk}(y))$;
return $y = y'$
Adversary $\mathcal{I}(x)$:
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$s \xleftarrow{\$} \{0,1\}^k$;
$c^\star \leftarrow x \,\|\, s$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
$y' \leftarrow [z \in L_H^{\mathcal{A}} \mid f_{pk}(z) = x]$;
return y'

$$\Pr_{\text{IND-CPA}(\mathcal{A})}[b' = b] - \tfrac{1}{2} \leq \Pr_{\text{OW}(\mathcal{I})}[y' = y]$$

Achievements

- Machine-Checked Proofs for Cryptographic Standards:
  - RSA: OAEP (PKCS#1 v2.x), PSS
  - CBC, CMAC, SHA3/Keccak
  - MAC-then-Encode-then-CBC-Encrypt (TLS-MEE-CBC)
  - most include low-level considerations
- Machine-Checked Proofs for Complex Constructions:
  - Yao's garbled circuits, semi-honest 2PC
  - Authenticated key exchange: PAKE, Naxos
  - Electronic voting: ballot privacy for 500+ variants of Helios
- Machine-Checked Proofs for Side-Channel Countermeasures:
  - Verified probing security of mask refreshing and multiplication gadgets

A language for cryptographic games

$\mathcal{C}$	::=	skip	skip
	|	$\mathcal{V} \leftarrow \mathcal{E}$	assignment
	|	$\mathcal{V} \xleftarrow{\$} \mathcal{D}$	random sampling
	|	$\mathcal{C};\ \mathcal{C}$	sequence
	|	if $\mathcal{E}$ then $\mathcal{C}$ else $\mathcal{C}$	conditional
	|	while $\mathcal{E}$ do $\mathcal{C}$	while loop
	|	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call

- $\mathcal{E}$: (higher-order) expressions
- $\mathcal{D}$: discrete sub-distributions
- $\mathcal{P}$: procedures

} user extensible

. oracles: concrete procedures

. adversaries: constrained abstract procedures

. parameters: for generic arguments

Reasoning about cryptographic games

- Probablistic Relational Hoare Logic (pRHL): program equivalences
- Probabilistic Hoare Logic (pHL): probability computations
- Hoare Logic (HL): safety properties
- Ambient logic. Program logic judgments are valid ambient logic formulas, allowing:
  - ☞ generic arguments;
  - ☞ modular proofs;
  - ☞ meta-arguments.

(Deductive) program verification

- Art of proving that programs are correct
- Origins:
  - ☞ axiomatic semantics (Hoare'69)
  - ☞ weakest precondition calculus (Floyd'67)
- Major advances in:
  - ☞ language coverage
  - ☞ automation
  - ☞ proof engineering

Hoare logic

- Judgments $c : P \Longrightarrow Q$
  (P and Q are first order formulae over program variables)
- A judgment $c : P \Longrightarrow Q$ is valid whenever

$$\forall m,\ m \models P \Rightarrow [\![c]\!]m = m' \Rightarrow m' \models Q$$

Selected rules

$$\frac{c : P \Longrightarrow Q \quad P' \Rightarrow P \quad Q \Rightarrow Q'}{c : P' \Longrightarrow Q'} \qquad \frac{}{x \leftarrow e : Q[e/x] \Longrightarrow Q}$$

$$\frac{c_1 : P \Longrightarrow Q \quad c_2 : Q \Longrightarrow R}{c_1 ; c_2 : P \Longrightarrow R}$$

$$\frac{c_1 : P \wedge e \Longrightarrow Q \quad c_2 : P \wedge \neg e \Longrightarrow Q}{\text{if } e \text{ then } c_1 \text{ else } c_2 : P \Longrightarrow Q}$$

$$\frac{c : I \wedge e \Longrightarrow I}{\text{while } e \text{ do } c : I \Longrightarrow I \wedge \neg e}$$

Verification condition generation

- Generate a set of verification conditions from annotated command and postcondition
- If all VCs are valid and $P \Rightarrow \mathrm{wp}(c, Q)$ then $c : P \Longrightarrow Q$

Selected rules

$$\begin{aligned}
\mathrm{wp}(x \leftarrow e, Q) &= Q[e/x] \\
\mathrm{wp}(c_1; c_2, R) &= \mathrm{wp}(c_1, \mathrm{wp}(c_2, R)) \\
\mathrm{wp}(\text{if } e \text{ then } c_1 \text{ else } c_2, Q) &= e \Rightarrow \mathrm{wp}(c_1, Q) \wedge \neg e \Rightarrow \mathrm{wp}(c_2, Q) \\
\mathrm{wp}(\text{while}_I\ e \text{ do } c, Q) &= I
\end{aligned}$$

The while rule generates two proof obligations

$$I \wedge e \Rightarrow \mathrm{wp}(c, I) \qquad I \wedge \neg e \Rightarrow Q$$

Beyond safety

2-safety and 2-programs safety

- 2 executions of the same program: information flow
- 2 programs: program equivalence

- Judgments

$$\vDash \{P\}\ c_1\ \sim\ c_2\ \{Q\}$$

(P and Q are f.o. formulae over tagged program variables)

- Validity

$$\forall m_1, m_2.\ (m_1, m_2) \vDash P \implies (\llbracket c_1 \rrbracket\ m_1, \llbracket c_2 \rrbracket\ m_2) \vDash Q$$

- Verification methods
  - ☞ embedding into Hoare logic
  - ☞ relational Hoare logic

pHL: probabilistic Hoare logic

- Judgment

  $$[c : P \Longrightarrow Q] \leq \delta \qquad [c : P \Longrightarrow Q] = \delta \qquad [c : P \Longrightarrow Q] \geq \delta$$

  where P and Q are predicates on memories, δ a real expression evaluated on initial memory
- Validity (intuition): Given an initial memory m such that P holds, the probability of Q in the distribution $[\![c_1]\!]\, m$ is related to δ (evaluated in m).

pRHL: probabilistic relational Hoare logic

- Judgment

$$\models \{P\}\ c_1\ \sim\ c_2\ \{Q\}$$

  where P and Q are relations on memories
- Validity (intuition): Given two initial memories m_1 and m_2 that are related by P, the distributions $[\![c_1]\!]\ m_1$ and $[\![c_2]\!]\ m_2$ are related by some lifting of Q to distributions.

Proving in EasyCrypt

- EasyCrypt's logic is a *general higher-order logic*.
- It comes with an interactive proof engine in which proofs are constructed.

---

lemma mylemma b1 b2 b3 :
(b1 $\Rightarrow$ b2) $\Rightarrow$ (b2 $\Rightarrow$ b3) $\Rightarrow$ b1 $\Rightarrow$ b3.
proof. *(∗ proof starts here ∗)*

$b1 : bool$
$b2 : bool$
$b3 : bool$
} local hypotheses (context)

$\underbrace{(b1 \Rightarrow b2) \Rightarrow (b2 \Rightarrow b3)}_{\text{assumptions}} \Rightarrow \underbrace{b1}_{\text{conclusion}} \Rightarrow b3$ } goal

Proving in EasyCrypt

- EasyCrypt's logic is a *general higher-order logic*.
- It comes with an interactive proof engine in which proofs are constructed.

---

lemma mylemma b1 b2 b3 :
(b1 $\Rightarrow$ b2) $\Rightarrow$ (b2 $\Rightarrow$ b3) $\Rightarrow$ b1 $\Rightarrow$ b3.
proof. *(∗ proof starts here ∗)*

$b1 : bool$
$b2 : bool$
$b3 : bool$
} local hypotheses (context)

$\underbrace{(b1 \Rightarrow b2) \Rightarrow (b2 \Rightarrow b3)}_{\text{assumptions}} \Rightarrow \underbrace{b1}_{\text{conclusion}} \Rightarrow b3$ } goal

Proving in EasyCrypt

- Progress is done via tactics that allows the *simplification*, *decomposition* into *subgoals*, or the *resolution* of the goal.

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ *b2*

(b2 ⇒ b3) ⇒ b1 ⇒ b3

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ b2
hb23 : b2 ⇒ b3
hb1 : b1

---

b3

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ b2
hb23 : b2 ⇒ b3
hb1 : b1

---

b2

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.
  apply hb12.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ b2
hb23 : b2 ⇒ b3
hb1 : b1

═══

b1

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.
  apply hb12.
  assumption.

Proof completed

Continuing the proof

```
lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.
  apply hb12.
  assumption.
qed.
```

Identification up to computations

EasyCrypt comes with a set of simplification rules.

$$\frac{a : \mathsf{bool} \quad b : \mathsf{bool}}{\mathsf{false} \mathbin{\hat{}\hat{}} b = (\mathsf{false} \wedge \neg b) \vee (\neg\mathsf{false} \wedge b)}$$

simplify leads to

$$\frac{a : \mathsf{bool} \quad b : \mathsf{bool}}{b = b}$$

that can be easily solved by reflexivity.

Identification up to computations

EasyCrypt comes with a set of simplification rules.

$$\frac{a : \mathsf{bool} \quad b : \mathsf{bool}}{\mathsf{false} \mathbin{\hat{}\hat{}} b = (\mathsf{false} \wedge \neg b) \vee (\neg\mathsf{false} \wedge b)}$$

simplify leads to

$$\frac{a : \mathsf{bool} \quad b : \mathsf{bool}}{b = b}$$

that can be easily solved by reflexivity.

Identification up to computations

Computations include

- reduction of functions applications
- operator inlining
- logical tautologies (a $\wedge$ false $\rightarrow$ false)

Terms that are equal up to computations are considered identical (in particular for the purpose of matching patterns for theorem application).

a : bool
b : bool

!a $\Rightarrow$ false ‘b = (false $\wedge$!b) $\vee$ (!false $\wedge$ b)

can be directly solved by reflexivity.

Tacticals

Tacticals are operators on tactics.

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time
  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied
- by t1; ...; tn
  apply t1; ...; tn and then try to close all the subgoals via trivial. fail if all the subgoals cannot be solved.

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time
  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied
- by t1; ...; tn
  apply t1; ...; tn and then try to close all the subgoals via trivial. fail if all the subgoals cannot be solved.

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time
  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied
- by t1; ...; tn
  apply t1; ...; tn and then try to close all the subgoals via trivial. fail if all the subgoals cannot be solved.

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time
  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied
- by t1; ...; tn
  apply t1; ...; tn and then try to close all the subgoals via trivial. fail if all the subgoals cannot be solved.

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time

  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied
- by t1; ...; tn
  apply t1; ...; tn and then try to close all the subgoals via trivial. fail if all the subgoals cannot be solved.

Tacticals

Tacticals are operators on tactics.

- t1; first t2

  apply t1 and then t2 on the first subgoal
- t1; last t2

  apply t1 and then t2 on the last subgoal
- variants: t1; first n t2, t1; last n t2
- t; first n last

  apply t and then shift the n first goals to the end

Tacticals

Tacticals are operators on tactics.

- t1; first t2

  apply t1 and then t2 on the first subgoal
- t1; last t2

  apply t1 and then t2 on the last subgoal
- variants: t1; first n t2, t1; last n t2
- t; first n last

  apply t and then shift the n first goals to the end

Tacticals - Intros

Tacticals are operators on tactics.

- t $\Rightarrow$ ip1 ... ipn

  apply t and then execute the introduction of ip1 ... ipn

  - t$\Rightarrow$ x

    introduce a name / an hypothesis
  - t$\Rightarrow$ [ip1|...ipn]

    execute ip_i on the i^{th} subgoal

    + do a case analysis if not done by t
  - t$\Rightarrow$ $\rightarrow$

    introduce an equational hypothesis and rewrite it
  - t$\Rightarrow$ {h}

    clear the hypothesis h
  - t$\Rightarrow$ //

    execute trivial

Tacticals - Intros

Tacticals are operators on tactics.

- t $\Rightarrow$ ip1 ... ipn

  apply t and then execute the introduction of ip1 ... ipn

  - t$\Rightarrow$ x

    introduce a name / an hypothesis
  - t$\Rightarrow$ [ip1|...ipn]

    execute ip_i on the i^{th} subgoal

    + do a case analysis if not done by t
  - t$\Rightarrow$ $\rightarrow$

    introduce an equational hypothesis and rewrite it
  - t$\Rightarrow$ {h}

    clear the hypothesis h
  - t$\Rightarrow$ //

    execute trivial