Introduction

July 16, 2013

Welcome

Acknowledgements:

- Benjamin Pierce, André Scedrov, U Penn support team
- Office of Naval Research
- EasyCrypt users

Organization:

- Lectures: overview of key components
- Labs: hands-on experience
- Workshop (Friday)

School web page:

http://www.easycrypt.info/school.html

EasyCrypt in a nusthell

- EasyCrypt is a tool-assisted platform for proving security of cryptographic constructions in the computational model
- Views cryptographic proofs as relational verification of open parametric probabilistic programs

- Leverage PL and PV techniques for cryptographic proofs
- Be accessible to cryptographers (choice of PL)
- Support high-level reasoning principles (still ongoing)
- Provide reasonable level of automation
- Reuse off-the-shelf verification tools (we use Why3)

EasyCrypt usage

- EasyCrypt is generic: no restriction on
  - ☞ primitives and protocols
  - ☞ security notions and assumptions
- Can be used interactively or as a certifying back-end
  - ☞ for cryptographic compilers (ZK)
  - ☞ for domain-specific (computational or symbolic) logics
- Can verify implementations
  - ☞ C-mode
  - ☞ CompCert as a certifying back-end

Evolution

Started in 2009. One older brother (CertiCrypt), started 2006.

- At first, mostly automated proofs
- v0.2 Interactive proofs in pRHL
- v1.0 Modular proofs, all layers explicit and with support for interactive proofs

Warning

v1.0 not yet finalized. Still needs to work on

- increasing automation
- high-level proof steps
- small(er) TCB
- ...

EasyCrypt: Languages

Typed imperative language

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

Expression language:

- features first-class distributions α *distr*
- allows higher-order expressions
- is extensible

Semantics of programs

Discrete sub-distribution transformers

$$[\![c]\!] : \mathcal{M} \to \mathcal{M} \text{ distr}$$

Probability of an event

$$\Pr[c, m : E] = [\![c]\!]_m \, E$$

Losslessness

$$\Pr[c, m : \top] = 1$$

EasyCrypt: Logics

- Ambient higher-order logic
- Hoare Logic $c : P \Longrightarrow Q$
- Probabilistic Hoare Logic (behind compute in v0.2)

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

- Probabilistic Relational Hoare Logic $c_1 \sim c_2 : P \Longrightarrow Q$

☞ Logics serve complementary purposes
☞ Some overlaps, many interplays
☞ HL, pHL, pRHL embedded in ambient logic

PRHL: intuition and preview

Judgment $c_1 \sim c_2 : P \Longrightarrow Q$ is valid iff for all memories m_1 and m_2

$$P\ m_1\ m_2 \Rightarrow Q^{\#}\ [\![c_1]\!]_{m_1}\ [\![c_2]\!]_{m_2}$$

Valid judgments allow deriving probability claims; eg if $P\ m_1\ m_2$ and $c_1 \sim c_2 : P \Longrightarrow Q$ and $Q \Rightarrow A\langle 1\rangle \Leftrightarrow B\langle 2\rangle$ then

$$\Pr[c_1, m_1 : A] = \Pr[c_2, m_2 : B]$$

Example rule:

$$\frac{c_1 \sim c : P \wedge e\langle 1\rangle \Longrightarrow Q \qquad c_2 \sim c : P \wedge \neg e\langle 1\rangle \Longrightarrow Q}{\text{if } e \text{ then } c_1 \text{ else } c_2 \sim c : P \Longrightarrow Q}$$

$$\frac{\begin{array}{c} P \Rightarrow e\langle 1\rangle = e'\langle 2\rangle \\ c_1 \sim c_1' : P \wedge e\langle 1\rangle \Longrightarrow Q \quad c_2 \sim c_2' : P \wedge \neg e\langle 1\rangle \Longrightarrow Q \end{array}}{\text{if } e \text{ then } c_1 \text{ else } c_2 \sim \text{if } e' \text{ then } c_1' \text{ else } c_2' : P \Longrightarrow Q}$$

EasyCrypt: modules and theories

Modules (beware memory model)

- Instantiating generic transformations (simplified syntax)

```
forall &m (A <: AdvCCA), exists (B <: AdvCPA),
      Pr[CCA(FO(S),A) @ &m : b' = b ] <=
      Pr[CPA(S,B) @ &m : b' = b] + ....
```

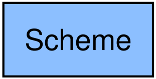
- Supporting high-level reasoning steps

Theories

- Supports code reuse
- “Polymorphism” via abstract types
- “Quantification” via abstract operators

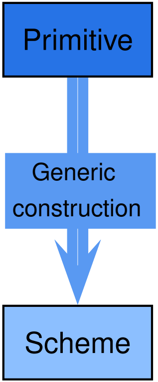
Plans to implement datatypes and type classes

Provable security

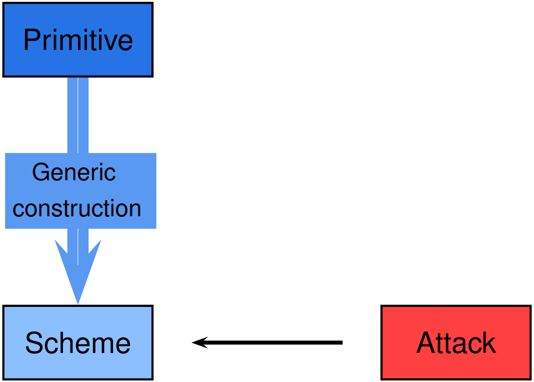


Provable security

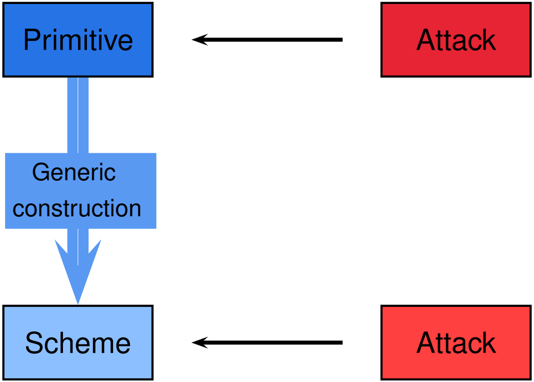
Provable security



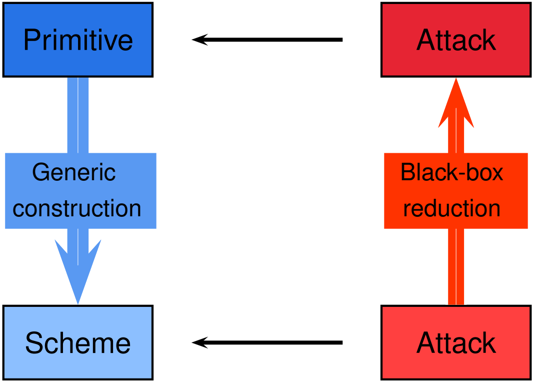
Provable security



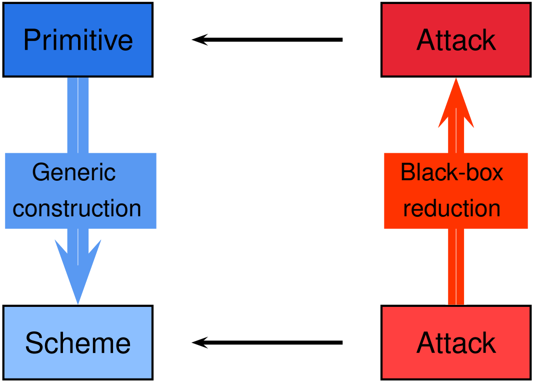
Provable security



Provable security



Provable security



Ideally attacks have similar execution times

Public-key encryption

Algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, s.t.:

- $\mathcal{E}$ takes as inputs a public key and a message, and outputs a ciphertext
- $\mathcal{D}$ takes as inputs a secret key and a ciphertext, and outputs a plaintext; $\mathcal{D}$ may be partial
- if (sk, pk) is a valid key pair, $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$

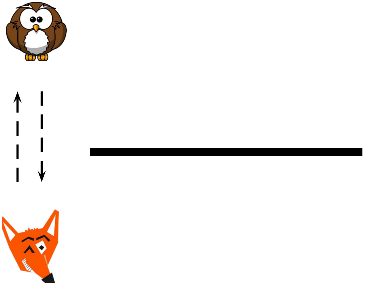
```
module type Scheme = {
  fun kg() : pkey * skey
  fun enc(pk:pkey, m:plaintext) : ciphertext
  fun dec(sk:skey, c:ciphertext) : plaintext option
}.
```

Correctness

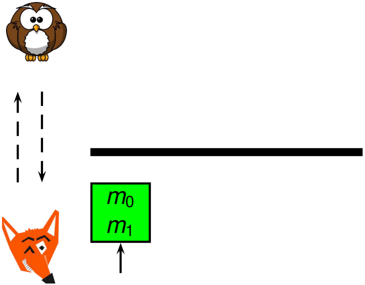
```
module Correct (S:Scheme) = {
  fun main(m:plaintext) : bool = {
    var pk : pkey;
    var sk : skey;
    var c : ciphertext;
    var m' : plaintext option;

    (pk, sk) = S.kg();
    c = S.enc(pk, m);
    m' = S.dec(sk, c);
    return (m' = Some m);
  }
}.
```

$$[Correctness(S, I) : \top \Longrightarrow \text{m'=Some m}] = 1$$

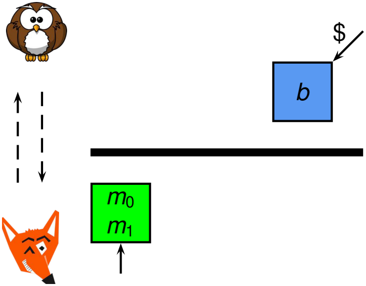
Indistinguishability



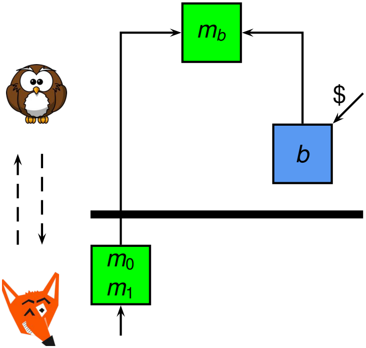
Indistinguishability



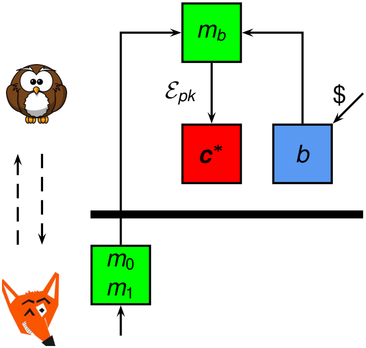
Indistinguishability



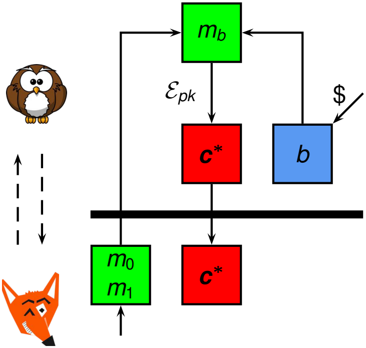
Indistinguishability



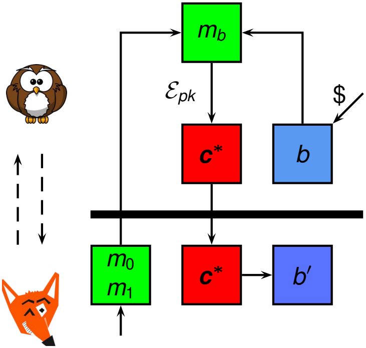
Indistinguishability



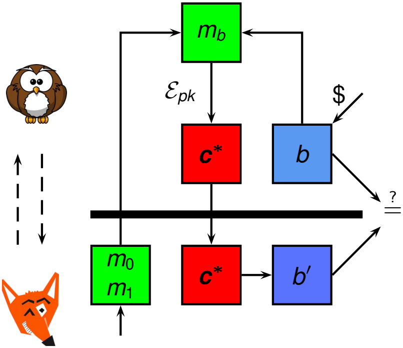
Indistinguishability



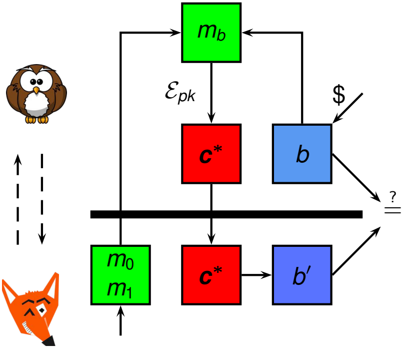
Indistinguishability



Indistinguishability



Indistinguishability



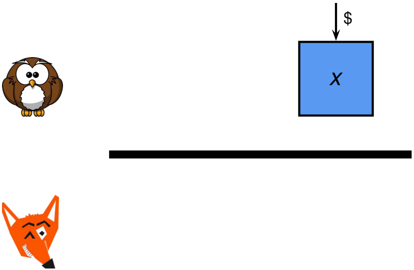
$$\left| \Pr\left[\text{IND-CCA}(\mathcal{A}) : b' = b \right] - \frac{1}{2} \right| \quad \text{small}$$

Indistinguishability

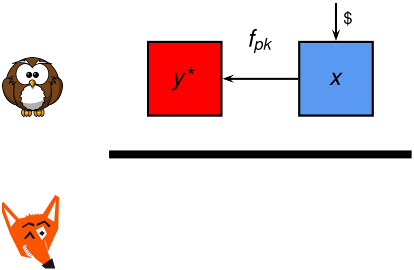
```
module CPA (S:Scheme, A:Adversary) = {
  fun main() : bool = {
    var pk : pkey;
    var sk : skey;
    var m0, m1 : plaintext;
    var c : ciphertext;
    var b, b' : bool;

    (pk, sk) = S.kg();
    (m0, m1) = A.choose(pk);
    b = ${0,1};
    c = S.enc(pk, b ? m1 : m0);
    b' = A.guess(c);
    return (b' = b);
  }
}.
```

One-way trapdoor permutations

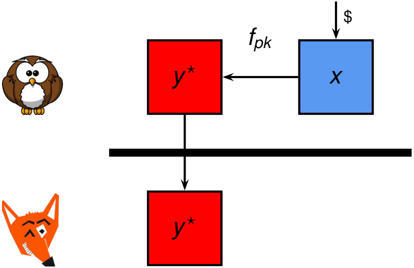
One-way trapdoor permutations



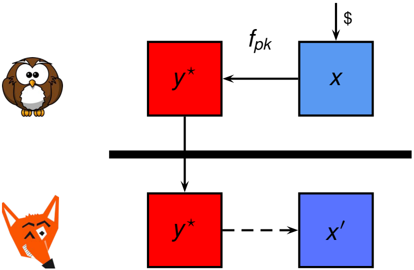
One-way trapdoor permutations



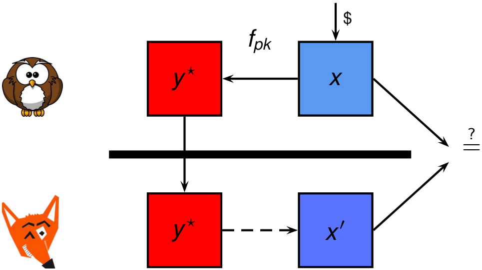
One-way trapdoor permutations



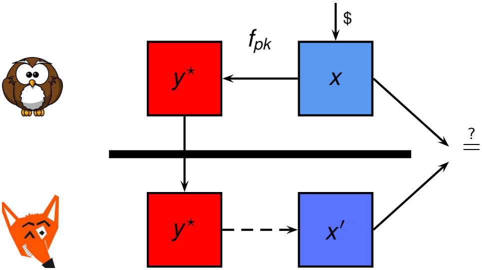
One-way trapdoor permutations



One-way trapdoor permutations



One-way trapdoor permutations



$$\Pr\left[\mathsf{OW}(\mathcal{I}) : x' = x\right] \quad \text{small}$$

One-way trapdoor permutations

```
module type Inverter = {
 fun i(pk : pkey, y : randomness) : randomness
}.

module OW(I :Inverter) ={
 fun main() : bool ={
 var x : randomness;
 var x' : randomness;
 var pk : pkey;
 var sk : skey;
   x = $uniform_rand;
   (pk,sk) = $keypairs;
   x' = I.i(pk,(f pk x));
   return (x' = x);
 }
}.
```

Random oracles (excerpts, and a bit of cheating)

```
module type Oracle =
{ fun init():unit
  fun o(x:from):to
}.

module type O_ext = { fun o(x:from):to }.

theory ROM.
  module RO:Oracle = {
    var m : (from, to) map

    fun o(x:from) : to = {
      var y : to;
      y = $dsample;
      if (!in_dom x m) m.[x] = y;
      return (m.[x]);
    }
  }.
```

Example: Bellare and Rogaway 1993 encryption

- plaintext is the type $\{0,1\}^n$ of bitstrings of length n
- randomness is the type $\{0,1\}^k$ of bitstrings of length k
- ciphertext is the type $\{0,1\}^{n+k}$ of bitstrings of length $n+k$

```
fun enc(pk:pkey, m:plaintext): ciphertext = {
  var h, s : plaintext;
  var r : randomness;

  r = ${0,1}^k;
  h = H.o(r);
  s = m ⊕ h;
  return ((f pk r) || s);
}
```

Security

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

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

Formal statement (omitting side conditions, simplified syntax)

```
forall &m (A <: Adv), exists (I <: Inverter),
    |Pr[CPA(BR,A).main() @ &m : b' = b ] − (1%r / 2%r)| <=
    Pr[OW(I).main() @ &m : x' = x].
```

Proof

Game hopping technique

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

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

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

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

1. For each hop
   - prove validity of pRHL judgment
   - derive probability claim(s)
2. Obtain security bound by combining claims
3. Check execution time of constructed adversary

Conditional equivalence

$$\begin{array}{l} \mathcal{E}_{pk}(m): \\ \quad r = \$\{0,1\}^{\ell}; \\ \quad h = H(r); \\ \quad s = h \oplus m; \\ \quad c = f_{pk}(r) \parallel s; \\ \quad \text{return } c; \end{array}
\quad \blacktriangleright \quad
\begin{array}{l} \mathcal{E}_{pk}(m): \\ \quad r = \$\{0,1\}^{\ell}; \\ \quad h = \$\{0,1\}^{k}; \\ \quad s = h \oplus m; \\ \quad c = f_{pk}(r) \parallel s; \\ \quad \text{return } c; \end{array}$$

$$\text{IND-CPA} \sim \mathbf{G} : \top \Longrightarrow (\neg r \in \boldsymbol{L}_H^{\mathcal{A}})\langle 2 \rangle \quad \Rightarrow \equiv$$

$$\left|\Pr\left[\text{IND-CPA} : b' = b\right] - \Pr\left[\mathbf{G} : b' = b\right]\right| \leq \Pr\left[\mathbf{G} : r \in \boldsymbol{L}_H^{\mathcal{A}}\right]$$

Equivalence

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

▶

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

$$\mathbf{G} \sim \mathbf{G}' : \top \Longrightarrow \equiv$$

$$\Pr\left[\mathbf{G} : r \in \boldsymbol{L}_H^{\mathcal{A}}\right] = \Pr\left[\mathbf{G}' : r \in \boldsymbol{L}_H^{\mathcal{A}}\right]$$
$$\Pr\left[\mathbf{G} : b' = b\right] = \Pr\left[\mathbf{G}' : b' = b\right] = \frac{1}{2}$$

Equivalence

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

►

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

$$\mathbf{G} \sim \mathbf{G}' : \top \Longrightarrow \equiv$$

$$\left|\Pr\left[\text{IND-CPA} : b' = b\right] - \tfrac{1}{2}\right| \leq \Pr\left[\mathbf{G}' : r \in \boldsymbol{L}_H^{\mathcal{A}}\right]$$

Reduction

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

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

$$\mathbf{G}' \sim \mathrm{OW} : \top \Longrightarrow (r \in \boldsymbol{L}_H^{\mathcal{A}})\langle 1 \rangle \Rightarrow (y' = y)\langle 2 \rangle$$

$$\Pr\left[\mathbf{G}' : r \in \boldsymbol{L}_H^{\mathcal{A}}\right] \leq \Pr\left[\mathrm{OW}(\mathcal{I}) : y' = y\right]$$

Reduction

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

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

$$\mathbf{G}' \sim \mathrm{OW} : \top \Longrightarrow (r \in \boldsymbol{L}_H^{\mathcal{A}})\langle 1 \rangle \Rightarrow (y' = y)\langle 2 \rangle$$

$$\left| \Pr\left[\text{IND-CPA}(\mathcal{A}) : b' = b\right] - \tfrac{1}{2} \right| \leq \Pr\left[\mathrm{OW}(\mathcal{I}) : y' = y\right]$$

Remarks

- In EasyCrypt v0.2, reasoning principles are "embedded " in pRHL proofs for the concrete construction
- In EasyCrypt v1, one can
  - ☞ prove high-level principles in an abstract setting
  - ☞ instantiate principles

  Benefits: much easier! Also favours
  - ☞ libraries of verified high-level principles
  - ☞ better proofs (shorter, faster, more robust)

Variations on IND-CPA

For every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ st

$$\left|\Pr\left[\text{IND-CPA}(\mathcal{A}) : b' = b\right] - \frac{1}{2}\right| = \Pr\left[\text{IND-CPA}(\mathcal{B}) : b' = b\right] - \frac{1}{2}$$

By case analysis on $\Pr[\text{IND-CPA}(\mathcal{A}) : b' = b] \leq \frac{1}{2}$

- If true, then $\mathcal{B}$ returns the result of $\mathcal{A}$
- If false, then $\mathcal{B}$ returns the negation of the result of $\mathcal{A}$

Summary

Provable security as deductive relational verification
of (open and parametrized) probabilistic programs

- EasyCrypt v1.0 is more explicit than its predecessor
- EasyCrypt v1.0 supports modular reasoning
- Shift of perspective (more instantiation, less pRHL)
- Should make tool more accessible to cryptographers