

IACR School on Computer-Aided Cryptography

Organizer: Benedikt Schmidt

Lecturers: F. Dupressoir, B. Schmidt, A. Stoughton

with help from: G. Barthe, B. Grégoire, P.-Y. Strub

and support from; the IACR, Jonathan Katz (local organizer)

Programme

Monday, June 1st

Morning (L)	Introduction to Game-Based and Computer-Aided Crypto Proofs
Afternoon (T)	Introduction to EasyCrypt Proofs <ul style="list-style-type: none">▶ Demo: Bellare & Rogaway▶ Exercise: a simple PRG▶ Exercise: Hashed ElGamal

Tuesday, June 2nd

Morning (T)	Modules and Instantiation <ul style="list-style-type: none">▶ Demo: Encrypt-then-MAC▶ Exercise: Encrypt-then-MAC▶ Exercise: The PRP-PRF switching lemma
Afternoon (T)	Reasoning up to Failure, Probability Computation <ul style="list-style-type: none">▶ Generic birthday bound, Encrypt-then-MAC, “Guessing Game”-style arguments

Programme (continued)

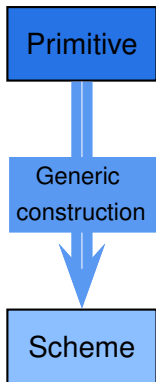
Wednesday, June 3rd

Morning (L)	EasyCrypt in practice <ul style="list-style-type: none">▶ Computer-Aided Cryptographic Proofs for Low-Level Implementations▶ EasyCrypt: Report on a case-study
Afternoon (T)	ZooCrypt: Automated Proofs for Pairing and RSA-Based Cryptography

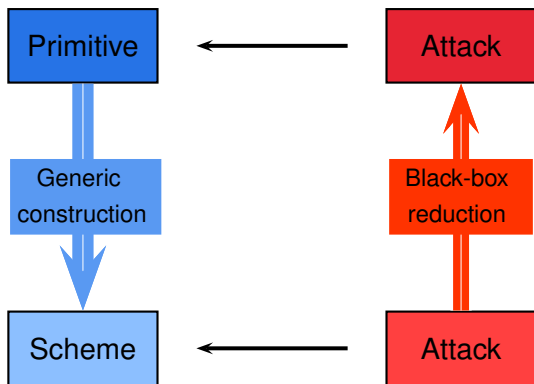
Thursday, June 4th

Morning (1)	<ul style="list-style-type: none">▶ Automated Analysis of Assumptions and Constructions in Generic Group Models▶ Automated Analysis and Synthesis of Modes of Operation for SKE and AE
Morning (2)	Open Slots <ul style="list-style-type: none">▶ Rene Peralta?

Reductionist proof



Reductionist proof



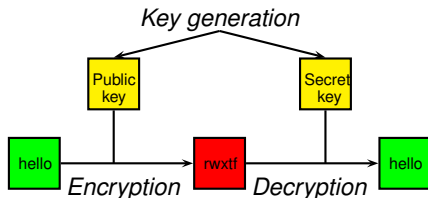
Public-key encryption

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})

$(sk, pk) \leftarrow \mathcal{K}();$

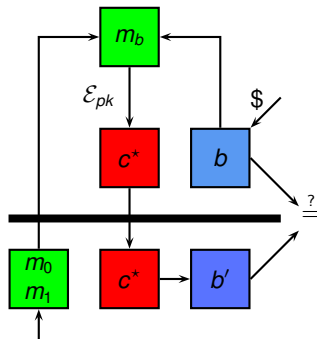
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$

$b \xleftarrow{\$} \{0, 1\};$

$c^* \leftarrow \mathcal{E}_{pk}(m_b);$

$b' \leftarrow \mathcal{A}_2(c^*);$

return $(b' = b)$



Indistinguishability

Game IND-CPA(\mathcal{A})

$(sk, pk) \leftarrow \mathcal{K}()$;

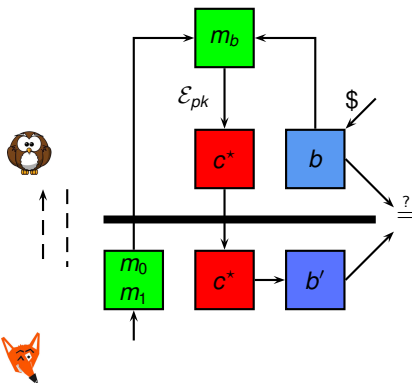
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;

$b \xleftarrow{\$} \{0, 1\}$;

$c^* \leftarrow \mathcal{E}_{pk}(m_b)$;

$b' \leftarrow \mathcal{A}_2(c^*)$;

return $(b' = b)$



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

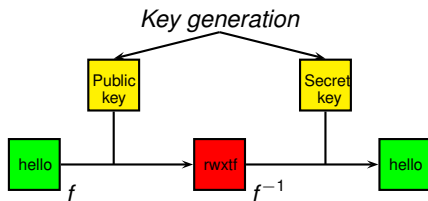
One-way trapdoor permutations

Algorithms $(\mathcal{K}, f_{pk}, f_{sk}^{-1})$

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

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

$$f_{sk}^{-1}(f_{pk}(m)) = m$$



One-way trapdoor permutations

Game $\text{OW}(\mathcal{I})$

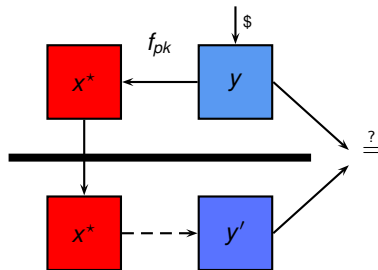
$(sk, pk) \leftarrow \mathcal{K}()$;

$y \xleftarrow{\$} \{0, 1\}^n$;

$x^* \leftarrow f_{pk}(y)$;

$y' \leftarrow \mathcal{I}(x^*)$;

return $(y' = y)$



One-way trapdoor permutations

Game $\text{OW}(\mathcal{I})$

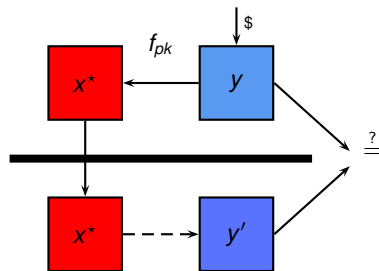
$(sk, pk) \leftarrow \mathcal{K}()$;

$y \xleftarrow{\$} \{0, 1\}^n$;

$x^* \leftarrow f_{pk}(y)$;

$y' \leftarrow \mathcal{I}(x^*)$;

return $(y' = y)$



$\Pr_{\text{OW}(\mathcal{I})}[y' = y]$ 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}(\);$

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$

$b \xleftarrow{\$} \{0, 1\};$

$c^* \leftarrow \mathcal{E}_{pk}(m_b);$

$b' \leftarrow \mathcal{A}_2(c^*);$

return $(b' = b)$

Encryption $\mathcal{E}_{pk}(m)$:

$r \xleftarrow{\$} \{0, 1\}^\ell;$

$s \leftarrow H(r) \oplus m;$

$y \leftarrow f_{pk}(r) \parallel s;$

return y

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

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

Proof

Game hopping technique

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

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

Game G' :
 $(sk, pk) \leftarrow \mathcal{K}();$
 $(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
 $b \xleftarrow{\$} \{0, 1\};$
 $c^* \leftarrow \mathcal{E}_{pk}(m_b);$
 $b' \leftarrow \mathcal{A}_2(c^*);$
return $(b' = b)$
Encryption $\mathcal{E}_{pk}(m)$:
 $r \xleftarrow{\$} \{0, 1\}^\ell;$
 $s \xleftarrow{\$} \{0, 1\}^k;$
 $h \leftarrow s \oplus m;$
 $c \leftarrow f_{pk}(r) \parallel s;$
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 $\mathcal{I}(x)$:
 $(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
 $s \xleftarrow{\$} \{0, 1\}^k;$
 $c^* \leftarrow x \parallel s;$
 $b' \leftarrow \mathcal{A}_2(c^*);$
 $y' \leftarrow [z \in L_H^A \mid f_{pk}(z) = x];$
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

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



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

By the Fundamental Lemma

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

Equivalence

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

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

By optimistic sampling

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

Equivalence

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



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

By optimistic sampling

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

Reduction

Game IND CPA :

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

Encryption $\mathcal{E}_{pk}(m)$:

$r \xleftarrow{\$} \{0, 1\}^\ell$;
 $s \xleftarrow{\$} \{0, 1\}^k$;
 $c \leftarrow f_{pk}(r) \parallel s$;
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 $\mathcal{I}(x)$:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
 $b \xleftarrow{\$} \{0, 1\}$;
 $s \xleftarrow{\$} \{0, 1\}^k$;
 $c^* \leftarrow x \parallel s$;
 $b' \leftarrow \mathcal{A}_2(c^*)$;
 $y' \leftarrow [z \in L_H^A \mid f_{pk}(z) = x]$;
return y'

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

Reduction

Game INDCPA :

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

Encryption $\mathcal{E}_{pk}(m)$:

$r \xleftarrow{\$} \{0, 1\}^\ell$;
 $s \xleftarrow{\$} \{0, 1\}^k$;
 $c \leftarrow f_{pk}(r) \parallel s$;
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 $\mathcal{I}(x)$:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
 $b \xleftarrow{\$} \{0, 1\}$;
 $s \xleftarrow{\$} \{0, 1\}^k$;
 $c^* \leftarrow x \parallel s$;
 $b' \leftarrow \mathcal{A}_2(c^*)$;
 $y' \leftarrow [z \in L_H^A \mid f_{pk}(z) = x]$;
return y'

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

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

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

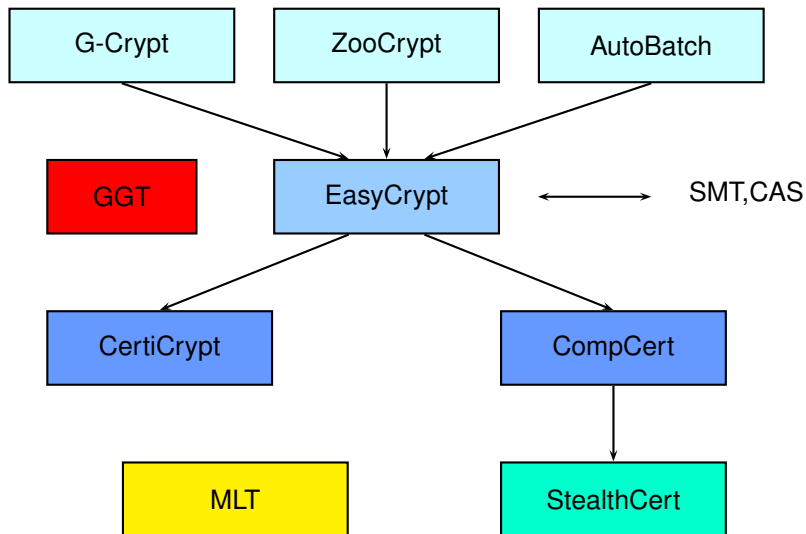
Real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security. Bernstein, 2013

Computer-aided cryptographic proofs

Provable security as deductive relational verification
of open probabilistic parameterized programs

- ▶ High-confidence reductionist 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

Overview of tools



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
	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \dots, \mathcal{E})$	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

- ▶ Probabilistic 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

$$\forall m, m' \models P \Rightarrow \llbracket c \rrbracket m = m' \Rightarrow m' \models Q$$

Selected rules

$$\frac{c : P \Longrightarrow Q \quad P' \Rightarrow P \quad Q \Rightarrow Q'}{c : P' \Longrightarrow Q'}$$

$$\frac{}{x \leftarrow e : Q[e/x] \Longrightarrow Q}$$

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

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

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

Verification condition generation

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

Selected rules

$$wp(x \leftarrow e, Q) = Q[e/x]$$

$$wp(c_1; c_2, R) = wp(c_1, wp(c_2, R))$$

$$wp(\text{if } e \text{ then } c_1 \text{ else } c_2, Q) = e \Rightarrow wp(c_1, Q) \wedge \neg e \Rightarrow wp(c_2, Q)$$

$$wp(\text{while}_I e \text{ do } c, Q) = I$$

The while rule generates two proof obligations

$$I \wedge e \Rightarrow wp(c, I) \quad I \wedge \neg e \Rightarrow Q$$

Beyond safety

2-safety and 2-programs safety

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

- ▶ Judgments

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

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

- ▶ Validity

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

- ▶ Verification methods

- ☞ embedding into Hoare logic
- ☞ relational Hoare logic

pHL: probabilistic Hoare logic

- ▶ Judgment

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

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 $\llbracket c_1 \rrbracket m$ 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 $\llbracket c_1 \rrbracket m_1$ and $\llbracket c_2 \rrbracket m_2$ 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 *)

$$\left. \begin{array}{l} b1 : bool \\ b2 : bool \\ b3 : bool \end{array} \right\} \text{local hypotheses (context)}$$

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

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 \Rightarrow hb12.

b1 : bool

b2 : bool

b3 : bool

hb12 : b1 \Rightarrow b2

(b2 \Rightarrow b3) \Rightarrow b1 \Rightarrow b3

Continuing the proof

lemma mylemma b1 b2 b3 : ...

proof.

move \Rightarrow hb12 bh23 hb1.

b1 : bool

b2 : bool

b3 : bool

hb12 : b1 \Rightarrow b2

hb23 : b2 \Rightarrow b3

hb1 : b1

b3

Continuing the proof

lemma mylemma b1 b2 b3 : ...

proof.

move \Rightarrow hb12 bh23 hb1.

apply hb23.

b1 : bool

b2 : bool

b3 : bool

hb12 : b1 \Rightarrow b2

hb23 : b2 \Rightarrow b3

hb1 : b1

b2

Continuing the proof

lemma mylemma b1 b2 b3 : ...

proof.

move \Rightarrow hb12 hb23 hb1.

apply hb23.

apply hb12.

b1 : bool

b2 : bool

b3 : bool

hb12 : b1 \Rightarrow b2

hb23 : b2 \Rightarrow b3

hb1 : b1

b1

Continuing the proof

lemma mylemma b1 b2 b3 : ...

proof.

move \Rightarrow hb12 bh23 hb1.

apply hb23.

apply hb12.

assumption.

Proof completed

Continuing the proof

lemma mylemma b1 b2 b3 : ...

proof.

move \Rightarrow hb12 bh23 hb1.

apply hb23.

apply hb12.

assumption.

qed.

Identification up to computations

EasyCrypt comes with a set of **simplification rules**.

a : bool

b : bool

$$\text{false} \wedge b = (\text{false} \wedge !b) \vee (!\text{false} \wedge b)$$

simplify leads to

a : bool

b : bool

$$b = b$$

that can be easily solved by **reflexivity**.

Identification up to computations

EasyCrypt comes with a set of **simplification rules**.

$$\frac{\begin{array}{l} a : \text{bool} \\ b : \text{bool} \end{array}}{\text{false} \wedge\wedge b = (\text{false} \wedge !b) \vee (!\text{false} \wedge b)}$$

simplify leads to

$$\frac{\begin{array}{l} a : \text{bool} \\ b : \text{bool} \end{array}}{b = b}$$

that can be easily solved by **reflexivity**.

Identification up to computations

Computations include

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

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

$$!a \Rightarrow \text{false} \wedge \wedge b = (\text{false} \wedge !b) \vee (!\text{false} \wedge b)$$

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 `ti` 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 be 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 `ti` 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 be 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 `ti` 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 be 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 be 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 be 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.

▶ $t \Rightarrow ip1 \dots ipn$

apply t and then execute the introduction of $ip1 \dots ipn$

● $t \Rightarrow x$

introduce a name / an hypothesis

● $t \Rightarrow [ip1 | \dots | ipn]$

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.

▶ $t \Rightarrow ip1 \dots ipn$

apply t and then execute the introduction of $ip1 \dots ipn$

• $t \Rightarrow x$

introduce a name / an hypothesis

• $t \Rightarrow [ip1 | \dots | ipn]$

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**

The rest of the week

- ▶ EasyCrypt tutorials
 - EasyCrypt forms a solid foundation for reasoning about cryptography and probabilistic programs;
- ▶ Lectures
 - Computer-aided cryptographic tools allow reasoning about more complex protocols, and more concrete aspects of cryptography;
- ▶ Talks
 - By us: some specific problems where automated solutions are forthcoming.
 - By you: anything you might have ready that relates to computer-aided cryptography.