

# Models and analysis of security protocols

## 1st Semester 2009-2010

### Link between Computational and Symbolic

### Introduction to Cryptoverif

## Lecture 12

Pascal Lafourcade

*Université Joseph Fourier, Verimag*

Master: November 16th 2009 thanks to Bruno Blanchet, Bruce Kapron and Steve Kremer

## Last Time (I)

### Lecture

- ▶ Non-interference
- ▶ Access Control

# Outline of Today

Link between Computational and Symbolic

Crypto Verif

Automated Cryptographic Proofs for Asymmetric Encryption

Conclusion

# Outline

Link between Computational and Symbolic

Crypto Verif

Automated Cryptographic Proofs for Asymmetric Encryption

Conclusion

# Messages

Computational world

Messages are bit-strings 11100010010111110010011

Symbolic world

Terms represent messages:  $\{\langle a, x, \rangle\}_k$

## Power of the Intruder

### Computational world

Adversary is represented by a PPTT machine.

### Symbolic world

Intruder is modeled by a deduction system, he has a limited power.  
Under the perfect encryption assumption.

# Proofs

## Computational world

Done by hand, long and often complex and hard to understand. (Sometime proofs are error-prone). Often based on reduction to cryptographic assumptions.

## Symbolic world

Based on constraint solving, First-order Logic, Tree automata etc ... these proofs are automatic.

## Abadi Rogaway 2000 Paper

- ▶ Introduced in [AR 00] M. Abadi et P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
- ▶ Models security of encryption against a *passive* computational adversary
- ▶ Adversary observes two *expressions* which have the same *pattern* assuming secure encryption – goal is to distinguish them with non-negligible probability

RESULT:

Symbolic indistinguishability implies computational security

## Patterns and Expressions

- ▶ *Patterns* are defined by the following grammar

$$\text{Keys} = \{k_1, k_2, \dots, \}$$

$$\text{Bool} = \{0, 1\}$$

$$P ::= k \in \text{Keys}$$

$$| b \in \text{Bool}$$

$$| (P_1, P_2)$$

$$| \{P\}_k$$

$$| \square$$

- ▶ An *expression* is a pattern with no occurrence of  $\square$
- ▶ e.g.,  $(\{\square\}_{k_2}, 1)_{k_1}$  is a pattern

## Key Recovery Function $\mathbf{F}_{kr}(E, K)$

- ▶ Following the approach of [Micciancio,Warinschi], for an expression  $E$  and a set  $K$  of keys, we define Key Recovery Function  $\mathbf{F}_{kr}(E, K)$  as follows:

$$\mathbf{F}_{kr}(b, K) = \emptyset$$

$$\mathbf{F}_{kr}(k, K) = \{k\} \cup K$$

$$\mathbf{F}_{kr}((E_1, E_2), K) = \mathbf{F}_{kr}(E_1, K) \cup \mathbf{F}_{kr}(E_2, K)$$

$$\mathbf{F}_{kr}(\{E\}_k, K) = \begin{cases} K & \text{if } k \notin K \\ \mathbf{F}_{kr}(E, K) & \text{otherwise.} \end{cases}$$

## Inductive Definition of Key Recovery **rec**

- ▶ Let  $E$  be an expression, then we define **rec** by  $\text{rec}(E) = \bigcup_i G_i(E) = G_{|E|}(E)$  where

$$G_0(E) = \emptyset$$

$$G_i(E) = \mathbf{F}_{kr}(E, G_{i-1}(E)).$$

- ▶  $\text{rec}(E)$  consists of the keys that can be recovered from  $E$  using information available in  $E$

### Exemple

$$E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$$

1.  $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$

## Inductive Definition of Key Recovery **rec**

- ▶ Let  $E$  be an expression, then we define **rec** by  $\text{rec}(E) = \bigcup_i G_i(E) = G_{|E|}(E)$  where

$$G_0(E) = \emptyset$$

$$G_i(E) = \mathbf{F}_{kr}(E, G_{i-1}(E)).$$

- ▶  $\text{rec}(E)$  consists of the keys that can be recovered from  $E$  using information available in  $E$

### Exemple

$$E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$$

1.  $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$
2.  $G_2(E) = \mathbf{F}_{kr}(E, \{k_1\}) = \{k_1, k_2\}$

## Inductive Definition of Key Recovery **rec**

- ▶ Let  $E$  be an expression, then we define **rec** by  $\text{rec}(E) = \bigcup_i G_i(E) = G_{|E|}(E)$  where

$$G_0(E) = \emptyset$$

$$G_i(E) = \mathbf{F}_{kr}(E, G_{i-1}(E)).$$

- ▶ **rec**( $E$ ) consists of the keys that can be recovered from  $E$  using information available in  $E$

### Exemple

$$E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$$

1.  $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$
2.  $G_2(E) = \mathbf{F}_{kr}(E, \{k_1\}) = \{k_1, k_2\}$
3.  $\text{rec}(E) = G_3(E) = G_4(E) = \{k_1, k_2, k_3\}$

## The pattern of an expression

- We denote by  $\mathbf{keys}(E)$  the set of all keys occurring in  $E$ , and let  $\mathbf{hidden}(E) = \mathbf{keys}(E) - \mathbf{rec}(E)$ . Define  $\mathbf{Pat}(E, K)$  as follows:

$$\mathbf{Pat}(b, K) = b$$

$$\mathbf{Pat}(k, K) = k$$

$$\mathbf{Pat}((E_1, E_2), K) = (\mathbf{Pat}(E_1, K), \mathbf{Pat}(E_2, K))$$

$$\mathbf{Pat}(\{E\}_k, K) = \begin{cases} \{\mathbf{Pat}(E, K)\}_k & \text{if } k \in K \\ \square & \text{otherwise} \end{cases}$$

Finally, we define  $\mathbf{pat}$  by  $\mathbf{pat}(E) = \mathbf{Pat}(E, \mathbf{rec}(E))$ . Intuitively,  $\mathbf{pat}(E)$  is the pattern corresponding to  $E$ , given the knowledge of any keys that may be recovered from  $E$ .

## Examples

$$\text{Pat}(E) = \text{Pat}(E, \text{rec}(E))$$

$$E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$$

$$\text{rec}(E) = \{k_1, k_2, k_3\}$$

$$\text{Pat}(E) = E$$

## Examples

$$\text{Pat}(E) = \text{Pat}(E, \text{rec}(E))$$

$$E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$$

$$\text{rec}(E) = \{k_1, k_2, k_3\}$$

$$\text{Pat}(E) = E$$

$$E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_2)$$

$$\text{rec}(E) = \{k_2, k_3\}$$

$$\text{Pat}(E) = (\square, \{k_3\}_{k_2}, k_2)$$

## Pattern Equivalence

- ▶ We write  $E \equiv F$  if  $\mathbf{pat}(E) = \mathbf{pat}(F)$
- ▶ We write  $E \cong F$  if there is a renaming of keys such that  $\mathbf{pat}(E) = \mathbf{pat}(F)$

Example:

- ▶  $K' \not\equiv K$  without renaming.
- ▶  $K' \cong K$  without renaming.
- ▶  $\langle k_1, k_2 \rangle \not\cong \langle k_3, k_3 \rangle$  (no bijection !)

# Pattern

## Exercise

$$\begin{aligned} \text{pat}(\langle \mathbf{0}, \mathbf{1} \rangle) &= ? \\ \text{pat}(\langle \langle \mathbf{0} \rangle_{K_1}, \langle \langle \mathbf{1} \rangle_{K_2}, K_1 \rangle \rangle) &= ? \\ \text{pat}(\langle \langle \langle \langle K_1 \rangle_{K_2} \rangle_{K_3}, K_3 \rangle \rangle) &= ? \end{aligned}$$

# Equivalence Examples

## Examples

$$0 \quad 0$$

$$0 \quad 1$$

$$\{0\}_K \quad \{1\}_K$$

$$\langle K, \{0\}_K \rangle \quad \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \quad \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \quad \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \quad \{0\}_K$$

$$\langle \{0\}_K, \{0\}_K \rangle \quad \langle \{0\}_K, \{1\}_K \rangle$$

# Equivalence Examples

## Examples

$$0 \approx 0$$

$$0 \quad 1$$

$$\{0\}_K \quad \{1\}_K$$

$$\langle K, \{0\}_K \rangle \quad \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \quad \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \quad \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \quad \{0\}_K$$

$$\langle \{0\}_K, \{0\}_K \rangle \quad \langle \{0\}_K, \{1\}_K \rangle$$

# Equivalence Examples

## Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

$$\{0\}_K \cong \{1\}_K$$

$$\langle K, \{0\}_K \rangle \cong \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \cong \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \cong \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \cong \{0\}_K$$

$$\langle \{0\}_K, \{0\}_K \rangle \cong \langle \{0\}_K, \{1\}_K \rangle$$

# Equivalence Examples

## Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

$$\{0\}_K \cong \{1\}_K$$

$$\langle K, \{0\}_K \rangle \quad \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \quad \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \quad \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \quad \{0\}_K$$

$$\langle \{0\}_K, \{0\}_K \rangle \quad \langle \{0\}_K, \{1\}_K \rangle$$

# Equivalence Examples

## Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

$$\{0\}_K \cong \{1\}_K$$

$$\langle K, \{0\}_K \rangle \not\cong \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \cong \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \cong \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \cong \{0\}_K$$

$$\langle \{0\}_K, \{0\}_K \rangle \cong \langle \{0\}_K, \{1\}_K \rangle$$

# Equivalence Examples

## Examples

$$\mathbf{0} \cong \mathbf{0}$$

$$\mathbf{0} \not\cong \mathbf{1}$$

$$\{\mathbf{0}\}_K \cong \{\mathbf{1}\}_K$$

$$\langle K, \{\mathbf{0}\}_K \rangle \not\cong \langle K, \{\mathbf{1}\}_K \rangle$$

$$\langle K, \{\langle \{\mathbf{0}\}_{K'}, \mathbf{0} \rangle\}_K \rangle \cong \langle K, \{\langle \{\mathbf{1}\}_{K'}, \mathbf{0} \rangle\}_K \rangle$$

$$\{\mathbf{0}\}_K \quad \{K\}_K$$

$$\{\langle \langle \mathbf{0}, \mathbf{0} \rangle \langle \mathbf{0}, \mathbf{0} \rangle \rangle\}_K \quad \{\mathbf{0}\}_K$$

$$\langle \{\mathbf{0}\}_K, \{\mathbf{0}\}_K \rangle \quad \langle \{\mathbf{0}\}_K, \{\mathbf{1}\}_K \rangle$$

# Equivalence Examples

## Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

$$\{0\}_K \cong \{1\}_K$$

$$\langle K, \{0\}_K \rangle \not\cong \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \cong \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \cong \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \cong \{0\}_K$$

$$\langle \{0\}_K, \{0\}_K \rangle \cong \langle \{0\}_K, \{1\}_K \rangle$$

# Equivalence Examples

## Examples

$$\mathbf{0} \cong \mathbf{0}$$

$$\mathbf{0} \not\cong \mathbf{1}$$

$$\{\mathbf{0}\}_K \cong \{\mathbf{1}\}_K$$

$$\langle K, \{\mathbf{0}\}_K \rangle \not\cong \langle K, \{\mathbf{1}\}_K \rangle$$

$$\langle K, \{\langle \{\mathbf{0}\}_{K'}, \mathbf{0} \rangle\}_K \rangle \cong \langle K, \{\langle \{\mathbf{1}\}_{K'}, \mathbf{0} \rangle\}_K \rangle$$

$$\{\mathbf{0}\}_K \cong \{K\}_K$$

$$\{\langle \langle \mathbf{0}, \mathbf{0} \rangle \langle \mathbf{0}, \mathbf{0} \rangle \rangle\}_K \cong \{\mathbf{0}\}_K$$

assuming encryption hides length

$$\langle \{\mathbf{0}\}_K, \{\mathbf{0}\}_K \rangle \quad \langle \{\mathbf{0}\}_K, \{\mathbf{1}\}_K \rangle$$

# Equivalence Examples

## Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

$$\{0\}_K \cong \{1\}_K$$

$$\langle K, \{0\}_K \rangle \not\cong \langle K, \{1\}_K \rangle$$

$$\langle K, \{\langle \{0\}_{K'}, 0 \rangle\}_K \rangle \cong \langle K, \{\langle \{1\}_{K'}, 0 \rangle\}_K \rangle$$

$$\{0\}_K \cong \{K\}_K$$

$$\{\langle \langle 0, 0 \rangle \langle 0, 0 \rangle \rangle\}_K \cong \{0\}_K$$

assuming encryption hides length

$$\langle \{0\}_K, \{0\}_K \rangle \cong \langle \{0\}_K, \{1\}_K \rangle$$

assuming encryption is probabilistic

# Encryption Cycles

## Definition by examples

- ▶  $\{k\}_k$
- ▶  $\{k_2\}_{k_1}, \{k_1\}_{k_2}$
- ▶  $\{k_2\}_{k_1}, \{k_3\}_{k_2}, \{k_1\}_{k_3}$
- ▶ etc ...

## Definition

For correctness result we need to avoid key cycle in terms.

### Definition (Key Cycles)

Let  $K, K' \in \mathbf{Keys}$ . We say that  $K$  crypts  $K'$  in the term  $M$ , denoted by  $K \succ_M K'$ , if  $\{N\}_K \in st(M)$  and  $K' \in st(N)$ .

A term  $M$  is **a**cyctic iff the relation  $\succ_M$  is acyclic.

## Definition

For correctness result we need to avoid key cycle in terms.

### Definition (Key Cycles)

Let  $K, K' \in \mathbf{Keys}$ . We say that  $K$  crypts  $K'$  in the term  $M$ , denoted by  $K \succ_M K'$ , if  $\{N\}_K \in st(M)$  and  $K' \in st(N)$ .

A term  $M$  is **acyclic** iff the relation  $\succ_M$  is acyclic.

### Examples

- ▶ Let  $M = \langle \{ \{ \{ K_1 \}_{K_2} \}_{K_3}, \mathbf{0} \rangle$ .  $\succ_M$  is defined by  $K_3 \succ_M K_2 \succ_M K_1$ .  $M$  is acyclic.

## Definition

For correctness result we need to avoid key cycle in terms.

### Definition (Key Cycles)

Let  $K, K' \in \mathbf{Keys}$ . We say that  $K$  crypts  $K'$  in the term  $M$ , denoted by  $K \succ_M K'$ , if  $\{N\}_K \in st(M)$  and  $K' \in st(N)$ .

A term  $M$  is **acyclic** iff the relation  $\succ_M$  is acyclic.

### Examples

- ▶ Let  $M = \langle \{ \{ \{ K_1 \}_{K_2} \}_{K_3}, \mathbf{0} \rangle$ .  $\succ_M$  is defined by  $K_3 \succ_M K_2 \succ_M K_1$ .  $M$  is acyclic.
- ▶ Let  $M = \{K\}_K$ . We get  $K \succ_M K$ .  $M$  has a cycle of size 1.

## Definition

For correctness result we need to avoid key cycle in terms.

### Definition (Key Cycles)

Let  $K, K' \in \mathbf{Keys}$ . We say that  $K$  crypts  $K'$  in the term  $M$ , denoted by  $K \succ_M K'$ , if  $\{N\}_K \in st(M)$  and  $K' \in st(N)$ .

A term  $M$  is **a**cyctic iff the relation  $\succ_M$  is acyclic.

### Examples

- ▶ Let  $M = \langle \{\{\{K_1\}_{K_2}\}_{K_3}, \mathbf{0}\} \rangle$ .  $\succ_M$  is defined by  $K_3 \succ_M K_2 \succ_M K_1$ .  $M$  is acyclic.
- ▶ Let  $M = \{K\}_K$ . We get  $K \succ_M K$ .  $M$  has a cycle of size 1.
- ▶ Let  $M = \langle \{\{K_1\}_{K_2}, \{K_2\}_{K_1}\} \rangle$ . We have that  $K_1 \succ_M K_2$  and  $K_2 \succ_M K_1$ .  $M$  has a cycle of size 2.

## Type-0 Scheme

- ▶ A *type-0* encryption scheme has the following properties:
  1. Repetition-hiding: given cipher-texts  $c, c'$ , cannot tell if corresponding plain-texts are equal (IND-CPA)
  2. Which-key-hiding: given cipher-texts  $c, c'$ , cannot tell if they were encrypted under the same key
  3. Message-length-hiding: given cipher-text  $c$ , cannot determine length of corresponding plain-text

## Type-0 Scheme

### Definition

Let  $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$  be an encryption scheme, and  $\mathcal{A}$  be an adversary:

$$\begin{aligned} Adv_{\Pi(n)}^0(\mathcal{A}) = & Pr[k, k' \leftarrow^R \mathcal{K}(n) : \mathcal{A}^{LR(\mathcal{E}_k(\cdot), \mathcal{E}_{k'}(\cdot))}(n) = 1] \\ & - Pr[k \leftarrow^R \mathcal{K}(n) : \mathcal{A}^{LR(\mathcal{E}_k(0), \mathcal{E}_k(0))}(n) = 1] \end{aligned}$$

$\Pi$  is Type-0 Scheme if  $Adv_{\Pi(n)}^0(\mathcal{A})$  is negligible.

Intuitively, an adversary can not say which encryption box is used only with input/output of encryption.

## Implementation of formal terms

Let  $\Pi$  an encryption scheme and  $\eta \in \text{Parameter}$ . We associate to the formal term  $M$  a distribution  $\llbracket M \rrbracket_{\Pi, \eta}$  and by consequence a family of distributions  $\llbracket M \rrbracket_{\Pi}$ .

**Initialize** $_{\eta}(M)$

for  $K \in \text{Keys}(M)$  do  $\tau(K) \stackrel{R}{\leftarrow} \mathcal{K}(\eta)$

**Convert** $(M)$

if  $M = K$  ( $K \in \mathbf{Keys}$ ) then return  $(\tau(K), \text{"key"})$

if  $M = b$  ( $b \in \mathbf{Bool}$ ) then return  $(b, \text{"bool"})$

if  $M = \langle M_1, M_2 \rangle$  then return  $(\mathbf{Convert}(M_1), \mathbf{Convert}(M_2), \text{"pair"})$

if  $M = \{M_1\}_K$  then

$x \stackrel{R}{\leftarrow} \mathbf{Convert}(M_1); \quad y \stackrel{R}{\leftarrow} \mathcal{E}_{\tau(K)}(x); \quad \text{return}(y, \text{"cipher-text"})$

## Implementation of formal terms

Let  $\Pi$  an encryption scheme and  $\eta \in \text{Parameter}$ . We associate to the formal term  $M$  a distribution  $\llbracket M \rrbracket_{\Pi, \eta}$  and by consequence a family of distributions  $\llbracket M \rrbracket_{\Pi}$ .

**Initialize $_{\eta}(M)$**

for  $K \in \text{Keys}(M)$  do  $\tau(K) \stackrel{R}{\leftarrow} \mathcal{K}(\eta)$

**Convert( $M$ )**

if  $M = K$  ( $K \in \mathbf{Keys}$ ) then return  $(\tau(K), \text{"key"})$

if  $M = b$  ( $b \in \mathbf{Bool}$ ) then return  $(b, \text{"bool"})$

if  $M = \langle M_1, M_2 \rangle$  then return  $(\mathbf{Convert}(M_1), \mathbf{Convert}(M_2), \text{"pair"})$

if  $M = \{M_1\}_K$  then

$x \stackrel{R}{\leftarrow} \mathbf{Convert}(M_1); \quad y \stackrel{R}{\leftarrow} \mathcal{E}_{\tau(K)}(x); \quad \text{return}(y, \text{"cipher-text"})$

- We generate all keys of  $M$  (denoted  $\text{Keys}(M)$ ) by calling  $\mathcal{K}$  and save it in a table  $\tau$

## Implementation of formal terms

Let  $\Pi$  an encryption scheme and  $\eta \in \text{Parameter}$ . We associate to the formal term  $M$  a distribution  $\llbracket M \rrbracket_{\Pi, \eta}$  and by consequence a family of distributions  $\llbracket M \rrbracket_{\Pi}$ .

**Initialize** $_{\eta}(M)$

for  $K \in \text{Keys}(M)$  do  $\tau(K) \stackrel{R}{\leftarrow} \mathcal{K}(\eta)$

**Convert** $(M)$

if  $M = K$  ( $K \in \mathbf{Keys}$ ) then return  $(\tau(K), \text{"key"})$

if  $M = b$  ( $b \in \mathbf{Bool}$ ) then return  $(b, \text{"bool"})$

if  $M = \langle M_1, M_2 \rangle$  then return  $(\mathbf{Convert}(M_1), \mathbf{Convert}(M_2), \text{"pair"})$

if  $M = \{M_1\}_K$  then

$x \stackrel{R}{\leftarrow} \mathbf{Convert}(M_1); \quad y \stackrel{R}{\leftarrow} \mathcal{E}_{\tau(K)}(x); \quad \text{return}(y, \text{"cipher-text"})$

- ▶ We generate all keys of  $M$  (denoted  $\text{Keys}(M)$ ) by calling  $\mathcal{K}$  and save it in a table  $\tau$
- ▶ We assume there is an implantation of constants  $\mathbf{0}$  and  $\mathbf{1}$

## Implementation of formal terms

Let  $\Pi$  an encryption scheme and  $\eta \in \text{Parameter}$ . We associate to the formal term  $M$  a distribution  $\llbracket M \rrbracket_{\Pi, \eta}$  and by consequence a family of distributions  $\llbracket M \rrbracket_{\Pi}$ .

**Initialize** $_{\eta}(M)$

for  $K \in \text{Keys}(M)$  do  $\tau(K) \xleftarrow{R} \mathcal{K}(\eta)$

**Convert** $(M)$

if  $M = K$  ( $K \in \mathbf{Keys}$ ) then return  $(\tau(K), \text{"key"})$

if  $M = b$  ( $b \in \mathbf{Bool}$ ) then return  $(b, \text{"bool"})$

if  $M = \langle M_1, M_2 \rangle$  then return  $(\mathbf{Convert}(M_1), \mathbf{Convert}(M_2), \text{"pair"})$

if  $M = \{M_1\}_K$  then

$x \xleftarrow{R} \mathbf{Convert}(M_1); y \xleftarrow{R} \mathcal{E}_{\tau(K)}(x);$  return  $(y, \text{"cipher-text"})$

- ▶ We generate all keys of  $M$  (denoted  $\text{Keys}(M)$ ) by calling  $\mathcal{K}$  and save it in a table  $\tau$
- ▶ We assume there is an implantation of constants  $\mathbf{0}$  and  $\mathbf{1}$
- ▶ We use tags to avoid ambiguity.

# Soundness

## Theorem

If  $E, F$  contain no encryption cycles using type-0 scheme and  $E \cong F$ , then  $\llbracket E \rrbracket \approx \llbracket F \rrbracket$

Proof is by a reduction-style argument.

## (Some) Extensions

- ▶ [Micciancio, Warinschi] give completeness for modified (authenticated) encryption, and also soundness for active adversaries
- ▶ [Micciancio, Panjwani] give a soundness theorem for an adversary which can adaptively attack the encryption scheme
- ▶ [Backes, Pfitzmann, Waidner] give a cryptographic implementation of Dolev-Yao terms in a general (UC) setting
- ▶ [Canneti, Herzog] give a formal (automated) approach to universal composability

# Outline

Link between Computational and Symbolic

**Crypto Verif**

Automated Cryptographic Proofs for Asymmetric Encryption

Conclusion

## An automatic prover doing Crypto Proofs

### Cryptoverif

- ▶ proves **secrecy** and **correspondence** properties.
- ▶ provides a **generic** method for specifying properties of **cryptographic primitives** which handles MACs (message authentication codes), symmetric encryption, public-key encryption, signatures, hash functions, . . .
- ▶ works for  **$N$  sessions** (polynomial in the security parameter), with an **active adversary**.
- ▶ gives a bound on the **probability** of an attack (exact security).

## Produced proofs

As in Shoup's method, the proof is a sequence of games:

- ▶ The first game is the **real protocol**.
- ▶ One goes from one game to the next by syntactic transformations or by applying the definition of security of a cryptographic primitive.  
The difference of probability between consecutive games is negligible.
- ▶ The last game is **"ideal"**: the security property can be read directly on it.  
(The advantage of the adversary is 0 for this game.)

## Process calculus for games

A game is formalized in a **process calculus**, essentially an **extension of the pi calculus**.

This calculus is inspired by:

- the calculus of [Lincoln, Mitchell, Mitchell, Scedrov]
- the calculus of [Laud, CCS'05]

The semantics is **purely probabilistic** (no non-determinism).

The runtime of processes is **polynomial in the security parameter**:

- polynomial number of copies of processes
- length of messages on channels bounded by polynomials

Extension to **arrays**.

## Process calculus for games: terms

$M ::=$	terms
$x, y, z, x[M_1, \dots, M_n]$	variable
$f(M_1, \dots, M_n)$	function application

Function symbols  $f$  correspond to functions computable by polynomial-time deterministic Turing machines.

## Process calculus for games: processes

$Q ::=$	input process
0	nil
$Q \mid Q'$	parallel composition
$!^{i \leq N} Q$	replication $N$ times
<b>newChannel</b> $c; Q$	restriction for channels
$c(x_1 : T_1, \dots, x_m : T_m); P$	input
$P ::=$	output process
$\bar{c}\langle M_1, \dots, M_m \rangle; Q$	output
<b>event</b> $e(M); P$	event
<b>new</b> $x : T; P$	random number generation (uniform)
$:= x : T MP$	assignment
<b>if</b> $M$ <b>then</b> $P$ <b>else</b> $P'$	conditional
<b>find</b> $j \leq N$ <b>suchthat</b> $\text{defined}(x[j], \dots) \wedge M$ <b>then</b> $P$ <b>else</b> $P'$	array lookup

# Arrays

Arrays replace **lists** often used in cryptographic proofs.

A variable defined under a replication is implicitly an **array**:

$$!^{i \leq N} \dots := x[i]M \dots$$

in fact defines  $x[i]$ , for  $i$  in  $1, \dots, N$ .

Under  $!^{i \leq N}$ , we write  $x$  for  $x[i]$ .

Requirements:

- ▶ Only variables with the current indexes can be assigned.
- ▶ Variables may be defined at several places, but only one definition can be executed for the same indexes.  
(if  $\dots$  then  $:= xMP$  else  $:= xM'P'$  is ok)

## Arrays (continued)

**find** performs an **array lookup**:

$$!^{i \leq N} \dots := xMP$$
$$| !^{i' \leq N'} c(y : T) \mathbf{find} \ j \leq N \ \mathbf{such\ that} \ \mathbf{defined}(x[j]) \wedge y = x[j] \ \mathbf{then} \ \dots$$

Note that **find** is here used outside the scope of  $x$ .

This is the only way of getting access to values of variables in other sessions.

When several array elements satisfy the condition of the **find**, the returned index is chosen randomly, with uniform probability.

## Main notion of security: observational equivalence

Two processes (games)  $Q_1, Q_2$  are **observationally equivalent** when the adversary has a negligible probability of distinguishing them:

$$Q_1 \approx Q_2$$

In the formal definition, the adversary is represented by an acceptable evaluation context

$C ::= [] \quad C \mid Q \quad Q \mid C \quad \mathbf{newChannel} \ c; C.$

Observational equivalence is an equivalence relation.

It is **contextual**:  $Q_1 \approx Q_2$  implies  $C[Q_1] \approx C[Q_2]$  where  $C$  is any acceptable evaluation context.

## Proof technique

We transform a game  $G_0$  into an observationally equivalent one using:

- ▶ **observational equivalences**  $L \approx R$  given as **axioms** and that come from security properties of primitives. These equivalences are used inside a context:

$$G_1 \approx_0 C[L] \approx C[R] \approx_0 G_2$$

- ▶ **syntactic transformations**: simplification, expansion of assignments, ...

We obtain a **sequence of games**  $G_0 \approx G_1 \approx \dots \approx G_m$ , which implies  $G_0 \approx G_m$ .

If some equivalence or trace property holds with overwhelming probability in  $G_m$ , then it also holds with overwhelming probability in  $G_0$ .

## MACs: security definition

A MAC takes as input a message and a secret key  $mac(m, k)$ .  
It comes with a checking function  $check$  such that

$$check(m, k, mac(m, k)) = true$$

A MAC guarantees the integrity and authenticity of the message because only someone who knows the secret key can build the mac.

More formally, an adversary  $\mathcal{A}$  that has oracle access to  $mac$  and  $check$  has a negligible probability to forge a MAC (UF-CMA):

$$\max_{\mathcal{A}} \Pr[check(m, k, t) \mid k \stackrel{R}{\leftarrow} mkgen; (m, t) \leftarrow \mathcal{A}^{mac(\cdot, k), check(\cdot, k, \cdot)}]$$

is negligible, when the adversary  $\mathcal{A}$  has not called the  $mac$  oracle on message  $m$ .

## MACs: intuitive implementation

By the previous definition, the adversary has a negligible probability of forging a correct MAC.

So when checking a MAC with  $check(m, k, t)$  and  $k$  is secret, the check can succeed **only if  $m$  is in the list (array) of messages whose  $mac$  has been computed** by the protocol.

So we can replace a check with an array lookup:  
if the call to  $mac$  is  $mac(x, k)$ , we replace  $check(m, k, t)$  with

**find  $j \leq N$  such that  $defined(x[j]) \wedge$   
 $(m = x[j]) \wedge check(m, k, t)$  then true else false**

Furthermore, we use primed function symbols after the transformation, so that it is not done again.

## MACs: formal implementation

$$\text{check}(m, \text{mkgen}(r), \text{mac}(m, \text{mkgen}(r))) = \mathbf{true}$$
$$\begin{aligned} & !^{N''} \mathbf{new} \ r : \text{mkeyseed}; ( \\ & \quad !^N (x : \text{bitstring}) \rightarrow \text{mac}(x, \text{mkgen}(r)), \\ & \quad !^{N'} (m : \text{bitstring}, t : \text{macstring}) \rightarrow \text{check}(m, \text{mkgen}(r), t)) \end{aligned}$$
$$\approx$$
$$\begin{aligned} & !^{N''} \mathbf{new} \ r : \text{mkeyseed}; ( \\ & \quad !^N (x : \text{bitstring}) \rightarrow \text{mac}'(x, \text{mkgen}'(r)), \\ & \quad !^{N'} (m : \text{bitsting}, t : \text{macstring}) \rightarrow \\ & \quad \quad \mathbf{find} \ j \leq N \ \mathbf{suchthat} \ \mathbf{defined}(x[j]) \wedge (m = x[j]) \wedge \\ & \quad \quad \quad \text{check}'(m, \text{mkgen}'(r), t) \ \mathbf{then} \ \mathbf{true} \ \mathbf{else} \ \mathbf{false}) \end{aligned}$$

The prover understands such specifications of primitives.

## MACs: formal implementation

The prover applies the previous rule automatically in **any (polynomial-time) context**, perhaps containing **several occurrences** of *mac* and or *check*:

- ▶ Each occurrence of *mac* is replaced with *mac'*.
- ▶ Each occurrence of *check* is replaced with a **find** that looks in all arrays of computed MACs (one array for each occurrence of function *mac*).

## Example

$$A \rightarrow B : \{x'_k\}_{x_k, x_{mk}}$$

where symmetric encryption is coded as encrypt-then-MAC.

$Q_0 = \text{start}(); \mathbf{new} \ x_r : \text{keyseed}; := x_k : \text{keykgen}(x_r)$   
 $\mathbf{new} \ x'_r : \text{mkeyseed}; := x_{mk} : \text{mkeymkgen}(x'_r) \overline{c} \langle \rangle; (Q_A \mid Q_B)$

$Q_A = !^{i' \leq n} c_A(); \mathbf{new} \ x'_k : \text{key}; \mathbf{new} \ x''_r : \text{coins};$   
 $:= x_m : \text{bitstringenc}(k2b(x'_k), x_k, x''_r)$   
 $\overline{c}_A \langle x_m, \text{mac}(x_m, x_{mk}) \rangle$

$Q_B = !^{i' \leq n} c_B(x'_m : \text{bitstring}, x_{ma} : \text{macstring});$   
 $\mathbf{if} \ \text{check}(x'_m, x_{mk}, x_{ma}) \ \mathbf{then}$   
 $:= i_{\perp}(k2b(x''_k)) \text{dec}(x'_m, x_k) \overline{c}_B \langle \rangle$

## Experiment Settings

Shared-key encryption is implemented as encrypt-then-MAC, using a IND-CPA encryption scheme.

(For Otway-Rees, we also considered a SPRP encryption scheme, a IND-CPA + INT-CTXT encryption scheme, a IND-CCA2 + IND-PTXT encryption scheme.)

Public-key encryption is assumed to be IND-CCA2.

## Experiments

Tested on the following protocols (original and corrected versions):

- ▶ Otway-Rees (shared-key)
- ▶ Yahalom (shared-key)
- ▶ Denning-Sacco (public-key)
- ▶ Woo-Lam shared-key and public-key
- ▶ Needham-Schroeder shared-key and public-key
- ▶ Full domain hash signature (with D. Pointcheval)
- ▶ Encryption schemes of Bellare-Rogaway'93 (with D. Pointcheval)

We prove secrecy of session keys and correspondence properties.

## Results

In most cases, the prover succeeds in proving the desired properties when they hold, and obviously it always fails to prove them when they do not hold.

Only cases in which the prover fails although the property holds:

- ▶ Needham-Schroeder public-key when the exchanged key is the nonce  $N_A$ .
- ▶ Needham-Schroeder shared-key: fails to prove that  $N_B[i] \neq N_B[i'] - 1$  with overwhelming probability, where  $N_B$  is a nonce
- ▶ Showing that the encryption scheme  $\mathcal{E}(m, r) = f(r) \| H(r) \oplus m \| H'(m, r)$  is IND-CCA2.

## Conclusion about CryptoVerif

Hopefully a promising approach.

Future extensions:

- ▶ Extension to **other cryptographic primitives**, in particular Diffie-Hellman.
- ▶ Improvements in the **proof strategy**.
- ▶ More **case studies**.

More information: <http://www.cryptoverif.ens.fr/>

B. Blanchet. A computationally sound mechanized prover for security protocols. In IEEE Symposium on Security and Privacy, pages 140–154, Oakland, California, May 2006. Extended version available as ePrint Report 2005/401.

# Outline

Link between Computational and Symbolic

Crypto Verif

**Automated Cryptographic Proofs for Asymmetric Encryption**

Conclusion

## Examples

- ▶ Bellare & Rogaway'93:

$$f(r)||x \oplus G(r)||H(x||r)$$

- ▶ Zheng & Seberry'93:

$$f(r)||G(r) \oplus (x||H(x))$$

- ▶ OAEP'94 (Bellare & Rogaway):

$$f(s||r \oplus H(s))$$

where  $s = x0^k \oplus G(r)$

- ▶ OAEP+'02 (Shoup):

$$f(s||r \oplus H(s))$$

where  $s = x \oplus G(r)||H'(r||x)$ .

- ▶ Fujisaki & Okamoto'99:

$$\mathcal{E}((x||r); H(x||r))$$

where  $\mathcal{E}$  is IND-CPA.

## Generic Encryption Scheme

Transform a one-way trapdoor permutation into an encryption schemes, using a simple programming language:

Command  $c ::= x \stackrel{r}{\leftarrow} \mathcal{U} \mid x := f(y) \mid x := f^{-1}(y)$   
 $\mid x := H(y) \mid x := y \oplus z$   
 $\mid x := y \parallel z \mid \text{if } x = y \text{ then } c_1 \text{ else } c_2 \text{ fi} \mid c; c$

Oracle declaration  $\mathcal{O} ::= \mathcal{N}(in, out) : body$

Where  $\mathcal{N}$  is the oracle's name,  $f$  one-way function.

# Generic Encryption Schemes

## Definition

A *generic encryption scheme* is given by a triple  $(\mathbb{F}, \mathcal{E}(\text{in}_e, \text{out}_e) : c, \mathcal{D}(\text{in}_d, \text{out}_d) : c')$ , where

1.  $\mathbb{F}$  is a *trapdoor permutation generator* that on input  $\eta$  generates an  $\eta$ -bit string trapdoor permutation  $(f, f^{-1})$  and
2.  $\mathcal{E}(\text{in}_e, \text{out}_e) : c$  and  $\mathcal{D}(\text{in}_d, \text{out}_d) : c'$  are oracle declarations.

## Example: Bellare-Rogaway'93

$$f(r) || in_e \oplus G(r) || H(in_e || r)$$

Encryption  $\mathcal{E}(in_e, out_e) =$

$r \xleftarrow{r} \mathcal{U};$

$a = f(r);$

$g := G(r);$

$b := in_e \oplus g;$

$t := in_e || r$

$c := H(t);$

$out_e := a || b || c$

Decryption  $\mathcal{D}(in_d, out_d)$

match  $in_d$  with  $a^* || b^* || c^*;$

$r^* := f^{-1}(a^*);$

$g^* := G(r^*);$

$m^* := b^* \oplus g^*;$

$t^* := m^* || r^*;$

$h^* := H(t^*);$

if  $h^* = c^*$  then  $out_d := m^*$

else  $out_d := error$

## Overview of the method for proving IND-CPA

Let  $GE = (\mathbb{F}, \mathcal{E}(\text{in}_e, \text{out}_e) : c, \mathcal{D}(\text{in}_d, \text{out}_d) : c')$  be a generic encryption scheme.

- ▶ Prove that  $\text{out}_e$  is indistinguishable from a randomly chosen value. ( $\Rightarrow$  IND-CPA)
- ▶ Assertion language:

$$\begin{aligned}\psi & ::= \text{Indis}(\nu x; V_1; V_2) \mid \text{WS}(x; V) \mid \text{H}(H, e) \\ \varphi & ::= \mathbf{true} \mid \psi \mid \varphi \wedge \varphi\end{aligned}$$

- ▶ Develop an analysis that automatically computes invariants:

$$\varphi \xrightarrow{c} \varphi'$$

## Three cornerstones: (1) Indis...

Two distributions  $X$  and  $X'$  are *indistinguishable* ( $X \sim X'$ ), if any adversary has a negligible advantage in distinguishing whether he is given a sample from  $X$  or  $X'$ .

- ▶ Indistinguishability, expressed by predicate `Indis` :

$$X \models \text{Indis}(\nu X, V_1, V_2)$$

$$X \sim_{V_1; V_2} [u \stackrel{r}{\leftarrow} \mathcal{U}; S \stackrel{r}{\leftarrow} X : S\{x \mapsto u\}]$$

- ▶ the adversary **sees**  $V_1$  and  $f(V_2)$  to help him choose.  
 $[S \stackrel{r}{\leftarrow} X : S(V_1), f(S(V_2))]$   $\sim$   
 $[S \stackrel{r}{\leftarrow} X ; u \stackrel{r}{\leftarrow} \mathcal{U}; S' := S\{x \mapsto u\} : (S'(V_1), f(S'(V_2)))]$

## Indistinguishability - Example

### Exemple

$$X = [x \stackrel{r}{\leftarrow} \{0, 1\}^n; y \stackrel{r}{\leftarrow} H(x)] \quad (1)$$

$$X' = [x \stackrel{r}{\leftarrow} \{0, 1\}^n; x' \stackrel{r}{\leftarrow} \{0, 1\}^n; y \stackrel{r}{\leftarrow} H(x')] \quad (2)$$

Then, we have  $X \sim_{\{y\};\{x\}} X'$  but we do not have  $X \sim_{\{y,x\}} X'$ .

$$X \models \text{Indis}(\nu y; y; x) \text{ but not } X \models \text{Indis}(\nu y; \{x, y\}; \emptyset).$$

Why?

## Indistinguishability - Example

### Exemple

$$X = [x \stackrel{r}{\leftarrow} \{0, 1\}^n; y \stackrel{r}{\leftarrow} H(x)] \quad (1)$$

$$X' = [x \stackrel{r}{\leftarrow} \{0, 1\}^n; x' \stackrel{r}{\leftarrow} \{0, 1\}^n; y \stackrel{r}{\leftarrow} H(x')] \quad (2)$$

Then, we have  $X \sim_{\{y\};\{x\}} X'$  but we do not have  $X \sim_{\{y,x\}} X'$ .

$$X \models \text{Indis}(\nu y; y; x) \text{ but not } X \models \text{Indis}(\nu y; \{x, y\}; \emptyset).$$

Why?

Because the adversary can query  $H$  on  $x$  and compare it to  $y$ !

## Three cornerstones: (2) $H(H, e) \dots$

- ▶ **ROM hypothesis:** Hash values must be asked for.
- ▶ Idea: the adversary cannot query hash oracles on randomized values.
- ▶ Not-Hashed-Yet, expressed by predicate  $H(H, e)$ :  
 $X \models H(H, e)$  iff  
 $\Pr[S \stackrel{r}{\leftarrow} X : S(e) \in \text{Arg}(H)]$  is negligible,  
where  $\text{Arg}(H)$  are arguments on which the hash oracle was queried.

## Three cornerstones : (3) WS...

- ▶ Many systems' security rely on the inability of an adversary to compute **the right argument** on which the oracle should be queried!
- ▶ Weak Secrecy, expressed by predicate WS:  
 $X \models \text{WS}(x; V)$  iff for any adversary  $\mathcal{A}$ ,  
 $\Pr[S \stackrel{r}{\leftarrow} X : \mathcal{A}(S(V)) = S(x)]$  is negligible.

## Summary of the Assertion language

$$\begin{aligned} \psi & ::= \text{Indis}(\nu x; V_1; V_2) \mid \text{WS}(x; V) \mid \text{H}(H, e) \\ \varphi & ::= \mathbf{true} \mid \psi \mid \varphi \wedge \varphi \end{aligned}$$

- ▶  $X \models \text{Indis}(\nu x : \mathcal{U}; V_1; V_2)$  iff  $X \sim_{V_1; V_2} [u \stackrel{r}{\leftarrow} \mathcal{U}; S \stackrel{r}{\leftarrow} X : S\{x \mapsto u\}]$ .
- ▶  $X \models \text{WS}(x; V)$  iff  $\Pr[S \stackrel{r}{\leftarrow} X : A(S) = S(x)]$  is negligible, for any adversary  $A$ .
- ▶  $X \models \text{H}(H, e)$  iff  $\Pr[S \stackrel{r}{\leftarrow} X : S(e) \in \text{Arg}(H)]$  is negligible.

...And A Few Rules...

►  $\{true\} x \stackrel{r}{\leftarrow} \mathcal{U} \{H(H, x)\},$

## ...And A Few Rules...

- ▶  $\{true\} x \stackrel{r}{\leftarrow} \mathcal{U} \{H(H, x)\},$
- ▶  $\{\text{Indis}(\nu y; V \cup \{y\}; \emptyset)\} x := f(y) \{\text{WS}(y; V \cup \{x\})\}$   
if  $y \notin V \cup \{x\}$

## ...And A Few Rules...

- ▶  $\{true\} x \stackrel{r}{\leftarrow} \mathcal{U} \{H(H, x)\},$
- ▶  $\{\text{Indis}(\nu y; V \cup \{y\}; \emptyset)\} x := f(y) \{WS(y; V \cup \{x\})\}$   
 if  $y \notin V \cup \{x\}$
- ▶  $\{WS(y; V) \wedge H(H, y)\} x := H(y) \{\text{Indis}(\nu x; V \cup \{x\}; \emptyset)\}$   
 etc...

Trapdoor permutation  $x := f(y)$

**Preservation rules:**

$$\begin{aligned} & \text{Indis}(\nu y; V; y) \longrightarrow \text{WS}(y; V, x) \quad (y \notin V \cup \{x\}) \\ & \text{Indis}(\nu y; V_1; V_2, y) \longrightarrow \text{Indis}(\nu x; V_1, x; V_2) \} \text{ provided } y \notin V_1 \cup V_2 \end{aligned}$$

Preservation rules: In the following rules, we assume  $z \neq x$ .

$$\begin{aligned} & \text{Indis}(\nu z; V_1, z; V_2, y) \longrightarrow \text{Indis}(\nu z; V_1, z, x; V_2) \} \text{ provided } z \neq y \\ & \text{H}(H, e) \longrightarrow \text{H}(H, e) \text{ provided } x \notin \text{var}(e) \\ & \text{WS}(z; V) \wedge \text{Indis}(\nu y; V, z; y) \longrightarrow \text{WS}(z; V, x) \end{aligned}$$

## Hash Function $x := H(y)$

Basic rules:

$$WS(y; V) \wedge H(H, y) \longrightarrow \text{Indis}(\nu x; V, x; )$$

$$WS(y; V) \wedge H(H, y) \longrightarrow H(H, e) \text{ provided } x \in \text{var}(e)$$

$$\text{Indis}(\nu y; V, y; V') \wedge H(H, y) \longrightarrow \text{Indis}(\nu x; V, x; V', y)$$

For both rules, we require  $y \notin V$ .

## Hash Function $x := H(y)$

Preservation rules: In the following rule, we assume  $x \neq y$  and  $z \neq x$ .

$$WS(y; V) \wedge WS(z; V) \wedge H(H, y) \longrightarrow WS(z; V, x)$$

$$H(H', e) \longrightarrow H(H', e) \text{ provided } H \neq H'$$

$$H(H, e) \wedge WS(z; y) \longrightarrow H(H, e) \text{ provided } z \in \text{var}(e)$$

$$\text{Indis}(\nu y; V_1, y; V_2) \wedge H(H, y) \longrightarrow \text{Indis}(\nu y; V_1, x; V_2, y) \\ \text{provided } y \notin P \cup V_1$$

$$\text{Indis}(\nu z; V_1, z; V_2) \wedge WS(y; V, z) \wedge H(H, y) \longrightarrow \text{Indis}(\nu z; V_1, x; V_2) \\ \text{provided } V_1 \cup V_2 \subseteq V$$

## Soundness of the analysis

### Prop

For every rule  $\{\varphi\}c\{\varphi'\}$ , we have

$$X \models \varphi \text{ implies } \llbracket c \rrbracket X \models \varphi'.$$

### Theorem

Let  $GE = (\mathbb{F}, \mathcal{E}(in_e, out_e) : c, \mathcal{D}(in_d, out_d) : c')$  be an asymmetric encryption scheme.

If  $\mathbf{true} \xrightarrow{c} \text{Indis}(\nu out_e; in_e, out_e; \emptyset)$  then  $\mathcal{E}$  is IND-CPA.

## Example

Bellare & Rogaway's 1993 generic construction.

$$\begin{array}{ll}
 r \xleftarrow{r} \{0, 1\}^{n_0} & - \quad \text{Indis}(\nu r) \wedge H(G, r) \wedge H(H, h||r) \\
 a := f(r) & - \quad \text{Indis}(\nu a; \text{Var} - r) \wedge \text{WS}(r; \text{Var} - r) \wedge \\
 & - \quad H(H, h||r) \\
 g := G(r) & - \quad \text{Indis}(\nu a; \text{Var} - r) \wedge \text{Indis}(\nu g; \text{Var} - r) \wedge \\
 & - \quad \text{WS}(r; \text{Var} - r) \wedge H(H, h||r) \\
 e := h \oplus g & - \quad \text{Indis}(\nu a; \text{Var} - r) \wedge \text{Indis}(\nu e) \wedge \\
 & - \quad \wedge \text{WS}(r; \text{Var} - r) \wedge H(H, h||r) \\
 d := h||r & - \quad \text{Indis}(\nu a) \wedge \text{Indis}(\nu e) \wedge \\
 & - \quad \text{WS}(r; \text{Var} - r) \wedge \\
 & - \quad H(H, d) \wedge \text{WS}(d) \\
 c := H(d) & - \quad \text{Indis}(\nu a) \wedge \text{Indis}(\nu e) \\
 & - \quad \wedge \text{Indis}(\nu c) \\
 \text{out}_e := a||e||c & - \quad \text{Indis}(\nu \text{oute}; \{\text{in}_e, \text{in}_d, \text{out}_e, \text{out}_d\})
 \end{array}$$

# Outline

Link between Computational and Symbolic

Crypto Verif

Automated Cryptographic Proofs for Asymmetric Encryption

Conclusion

# Summary

## Today

- ▶ Link Symbolic and Computational
- ▶ Pattern
- ▶ Key cycles
- ▶ Crypto Verif
- ▶ Automatic Verification for Generic Asymmetric Schemes  
(CCS 08 Courant Daubignard Ene Lafourcade Lacknech)

## General Summary

### Computational World

- ▶ Indistinguishability (IND/NM CPA/CCA1/CCA2)
- ▶ Reduction Proof, Hybrid Argument
- ▶ Cryptoverif
- ▶ Automatic procedure for assymmetric encryption

### Link between two worlds

### Symbolic World

- ▶ DY Passive/Active Intruder (Tools)
- ▶ Access Control, Non Interference
- ▶ Secret sharing, Side Channels, ZKP

## Next

- ▶ Exam 3rd December 2009
- ▶ 1h30
- ▶ 1 RV A4 of your notes
- ▶ All the topics

Thank you for your attention



Questions ?