

Advanced Cryptography 1st Semester 2007-2008

Link between Computational and Symbolic Introduction to Cryptoverif and others

Pascal Lafourcade

Université Joseph Fourier, Verimag

Master: December 1th 2008 thanks to Bruno Blanchet, Bruce Kapron and Steve Kremer

Last Time (I)

Lecture

- Acces Control

Outline of Today

- 1 Link between Computational and Symbolic
- 2 Crypto Verif
- 3 E-Cash
- 4 Funny Introduction to Zero Knowledge Proof
 - Principle
 - Funny example: Rubik's Cube
 - Example: Graph Coloring
- 5 Secret Sharing
- 6 Conclusion

Outline

- 1 Link between Computational and Symbolic
- 2 Crypto Verif
- 3 E-Cash
- 4 Funny Introduction to Zero Knowledge Proof
 - Principle
 - Funny example: Rubik's Cube
 - Example: Graph Coloring
- 5 Secret Sharing
- 6 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 proof are error-prone). Often based on reduction to cryptographic assumptions.

Symbolic world

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

Abadi Rogaway 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

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

$$G_0(E) = \emptyset$$

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

- rec**(E) consists of the keys that can be recovered from E using information available in E
- e.g., if $E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$, then
 - $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$

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

$$G_0(E) = \emptyset$$

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

- rec**(E) consists of the keys that can be recovered from E using information available in E
- e.g., if $E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$, then
 - $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$
 - $G_2(E) = \mathbf{F}_{kr}(E, \{k_1\}) = \{k_1, k_2\}$

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
- e.g., if $E = ((\{\{k_2\}_{k_1}\}_{k_1}, \{k_3\}_{k_2}), k_1)$, then
 - $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$
 - $G_2(E) = \mathbf{F}_{kr}(E, \{k_1\}) = \{k_1, k_2\}$
 - $\text{rec}(E) = G_3(E) = G_4(E) = \{k_1, k_2, k_3\}$

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

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

Equivalence Examples

Examples

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

Equivalence Examples

Examples

$$0 \cong 0$$

$$0 \quad 1$$

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

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

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

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

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

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

Equivalence Examples

Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

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

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

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

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

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

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

Equivalence Examples

Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

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

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

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

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

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

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

Equivalence Examples

Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

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

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

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

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

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

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

Equivalence Examples

Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

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

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

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

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

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

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

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

$$0 \cong 0$$

$$0 \not\cong 1$$

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

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

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

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

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

assuming encryption hides length

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

Equivalence Examples

Examples

$$0 \cong 0$$

$$0 \not\cong 1$$

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

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

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

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

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

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 **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 acyclique.

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 acyclique.
- 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 acyclique.
- 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 ciphertexts c, c' , cannot tell if corresponding plaintexts are equal (IND-CPA)
 - ② Which-key-hiding: given ciphertexts c, c' , cannot tell if they were encrypted under the same key
 - ③ Message-length-hiding: given ciphertext c , cannot determine length of corresponding plaintext

Type-0 Scheme

Definition

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

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

Π 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.

Implementantation of formal terms

Let Π 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

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

Implementation of formal terms

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

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

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

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

Implementation of formal terms

Let Π 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

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

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

Implementantation of formal terms

Let Π 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 $(y, \text{"ciphertext"})$

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

- 1 Link between Computational and Symbolic
- 2** Crypto Verif
- 3 E-Cash
- 4 Funny Introduction to Zero Knowledge Proof
Principle
Funny example: Rubik's Cube
Example: Graph Coloring
- 5 Secret Sharing
- 6 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
newChannel $c; Q$	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
event $e(M); P$	event
new $x : T; P$	random number generation (uniform)
$:= x : T MP$	assignment
if M then P else P'	conditional
find $j \leq N$ such that $\text{defined}(x[j], \dots) \wedge M$ then P else 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):

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

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  $j \leq N$  suchthat  $defined(x[j]) \wedge$   
 $(m = x[j]) \wedge check(m, k, t)$  then true else false
```

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

MACs: formal implementation

$$\text{check}(m, \text{mkgen}(r), \text{mac}(m, \text{mkgen}(r))) = \mathbf{true}$$

$$\begin{aligned} & !^{N''} \mathbf{new} \ r : \text{mkeyseed}; (\\ & \quad !^N (x : \text{bitstring}) \rightarrow \text{mac}(x, \text{mkgen}(r)), \\ & \quad !^{N'} (m : \text{bitstring}, t : \text{macstring}) \rightarrow \text{check}(m, \text{mkgen}(r), t)) \end{aligned}$$

$$\approx$$

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

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

- 1 Link between Computational and Symbolic
- 2 Crypto Verif
- 3 E-Cash**
- 4 Funny Introduction to Zero Knowledge Proof
 - Principle
 - Funny example: Rubik's Cube
 - Example: Graph Coloring
- 5 Secret Sharing
- 6 Conclusion

Notions

Payment ONLINE: Need confirmation of the bank

Payment OFFLINE: Confirmation not needed

Warnings

- Bank code
- integrity
- non-repudiation
- authentication
- privacy

Historic

- Credit cards over SSL (Paypal mostly used in US)
- e-cheques (Netcash 1993)
- Virtual credit cards (First Virtual 1994)
- Encrypted credit cards (Cybercash 1994)
- Mondex/SET – > Chip-SET (C-SET)
- EMV: Europay, Mastercard, Visa 1994
- Many, many others ... (Digicash 1994, Geldkarte)

SET or CSET

Used for Visa and MAsterCard 1997 with 3 principals:

- Customer (C)
- Merchandizer (M)
- Bank (B)

Properties Aimed

- Mutual authentication between M and C
- Authentication of B by C and M
- Secrecy of the command between C and B
- Secrecy of modalities of the payment between C and M
- Non repudiation of the transaction of the 3 principals
- Freshness of the transaction

Anonymity: B should not know that C bought something to M

Fairness: C should sign before M

SET: Secure Electronic Transaction

Uses RSA, DES and SHA1.

Card number known by C and B

Number and price of the command known by C, M and B

Command known by C and M

Notations

- N_M and N_C nonces
- Od = Oder details
- Pd = Payment details
- $Trans = C, M, (N_C, N_M), Price, hash(Od), hash(Pd)$

SET

$$C \rightarrow B : C, M$$

$$M \rightarrow C : N_M$$

$$C \rightarrow M : \{Trans\}_{K_C^{-1}}, \{Od\}_{K_M}, \{Pd\}_{K_B}$$

$$M \rightarrow B : \{Trans\}_{K_C^{-1}}, \{Trans\}_{K_M^{-1}}, \{Pd\}_{K_B}$$

$$B \rightarrow M : \{Results, hash(Trans)\}_{K_B^{-1}}$$

$$M \rightarrow C : \{Results, hash(Trans)\}_{K_B^{-1}}$$

SET

Claimed properties:

- Od is a shared secret between C and M .
- Pd is a shared secret between C and B .
- Mutual authentication between C and M with N_M and N_C
- Authentication of B by C and M with $hash(Trans)$
- Authentication of all messages by the 3 principals.
- No-repudiation due to signed and hashed $Trans$ by all principals.
- Freshness of the transaction checked by B .

Not achieved properties

- No anonymity: B knows that C buys something at M
- No Fairness S signs before M .

Outline

- 1 Link between Computational and Symbolic
- 2 Crypto Verif
- 3 E-Cash
- 4 Funny Introduction to Zero Knowledge Proof**
 - Principle
 - Funny example: Rubik's Cube
 - Example: Graph Coloring
- 5 Secret Sharing
- 6 Conclusion

Interactive Zero Knowledge Proofs

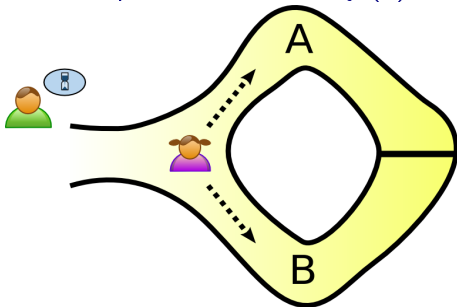
In an interactive zero knowledge proof, a prover P interacts with a verifier V to demonstrate the validity of an assertion without revealing anything about the assertion to the verifier.

An Example: The Cave Story (1)

- a cave shaped a circle
- a magic door at one side
- an entrance at the other side
- Victor will pay Peggy only if she knows the secret
- Peggy won't tell the secret until she would have been paid.

Interactive Zero Knowledge Proofs

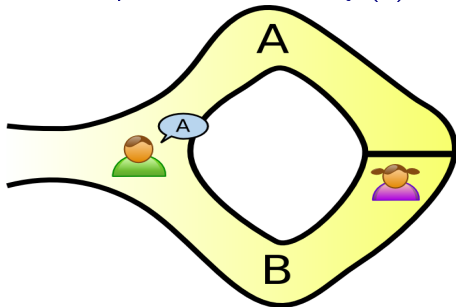
An Example: The Cave Story (2)



First, Victor waits outside while Peggy chooses a path.

Interactive Zero Knowledge Proofs

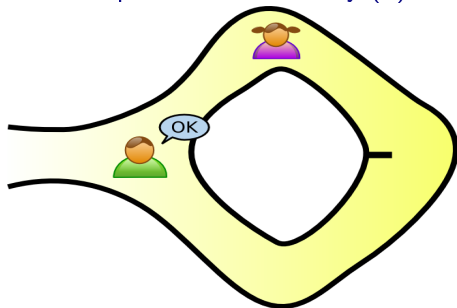
An Example: The Cave Story (3)



Then Victor enters and shouts the name of a path.

Interactive Zero Knowledge Proofs

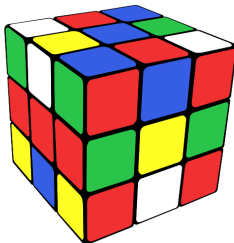
An Example: The Cave Story (4)



At last, Peggy returns along the desired path (using the secret if necessary).

Rubik's Cube

Assume Alice knows how to solve the Rubik's Cube.

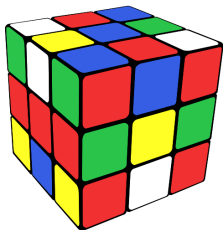


Question ?

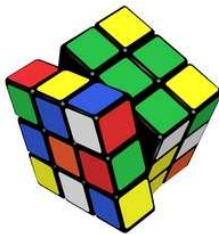
She wants to prove to Bob she has the skill to solved a given scrambled Rubik's cube without revealing it. How can she do it?

Rubik's Cube

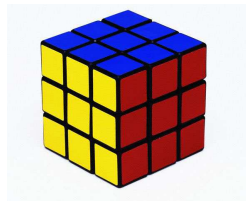
Alice scrambles the cube and proposed a new cube to Bob



1



2



3

Bob asks Alice from the current position (2) to go back to the initial position (1) or to solve it (3).

Repeating this process k times Bob is convinced that Alice knows the secret with a probability $1/2^k$.

Graph 3-coloring

Definition

3-coloring A 3-coloring of a graph is an assignment of 3 colors to vertices such that no pair of adjacent vertices are assigned to the same color.

3-coloring Problem

Given a graph G , the problem of deciding if the graph G is 3-colorable is an NP problem. (cf Garey and Johnson Book)

Problem: Alice wants to prove to Bob she knows a 3-coloring c of a given graph G .

Graph 3-coloring

- 1 Alice chooses a permutation π of the 3 colors ($\pi \circ c$ is still a 3-coloring of the graph G). And she transmits to Bob $e_u = H(\pi(c(u)) || r_u)$ for $u \in V$ and r_u random value.
- 2 Bob asks colors for u and v in V .
- 3 Alice answers $r_u, r_v, \pi(c(u)), \pi(c(u)), \pi(c(v))$ which allows Bob to confirm messages send by Alice.

Given a permutation π

$A \rightarrow B : \forall u \in V, e_u = H(\pi(c(u)) || r_u)$

$B \rightarrow A : u, v$

$A \rightarrow B : r_u, r_v, \pi(c(u)), \pi(c(u)), \pi(c(v))$

Graph 3-coloring

Playing several time this procedure Bob is convinced that Alice has a 3-coloring of G .

Completeness

If Alice knows the coloring then Bob will accept her proof.

Soundness

If Alice does not know the coloring then Bob will detect it with probability $\frac{1}{\#edges}$

Zero-knowledge

Bob just sees two random colors. Hence he learns nothing about the 3-coloring of G .

RK : ZKP are sensible to Man in the middle attack.

Outline

- 1 Link between Computational and Symbolic
- 2 Crypto Verif
- 3 E-Cash
- 4 Funny Introduction to Zero Knowledge Proof
 - Principle
 - Funny example: Rubik's Cube
 - Example: Graph Coloring
- 5 Secret Sharing**
- 6 Conclusion

Shamir 1979

Initial Problem

Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present.

- What is the smallest number of locks needed?
- What is the smallest number of keys to the locks each scientist must carry?

Shamir 1979

The minimal solution uses 462 locks and 252 keys per scientist.

Integrity

The lock can be opened with $m = 6$ parts over $n = 11$.

Confidentiality

No way to open the lock with less than $m = 6$ parts over $n = 11$.

Shamir 1979

What is the smallest number of locks needed?

Shamir 1979

What is the smallest number of locks needed?

Idea : For each group of five scientists, there must be at least one lock for which they do not have the key, but for which every other scientist does have the key.

There are $\binom{11}{5} = 462$ groups of five scientists, there must be at least 462 locks.

Shamir 1979

What is the smallest number of locks needed?

Idea : For each group of five scientists, there must be at least one lock for which they do not have the key, but for which every other scientist does have the key.

There are $\binom{11}{5} = 462$ groups of five scientists, there must be at least 462 locks.

What is the smallest number of keys to the locks each scientist must carry?

Shamir 1979

What is the smallest number of locks needed?

Idea : For each group of five scientists, there must be at least one lock for which they do not have the key, but for which every other scientist does have the key.

There are $\binom{11}{5} = 462$ groups of five scientists, there must be at least 462 locks.

What is the smallest number of keys to the locks each scientist must carry?

Similarly, each scientist must hold at least one key for every group of five scientists of which they are not a member, and there are $\binom{10}{5} = 252$ such group

Shamir 1979

What is the smallest number of locks needed?

Idea : For each group of five scientists, there must be at least one lock for which they do not have the key, but for which every other scientist does have the key.

There are $\binom{11}{5} = 462$ groups of five scientists, there must be at least 462 locks.

What is the smallest number of keys to the locks each scientist must carry?

Similarly, each scientist must hold at least one key for every group of five scientists of which they are not a member, and there are $\binom{10}{5} = 252$ such group

If we generalize we get $\binom{n}{m-1}$ and $\binom{n-1}{m-1}$.

Secret Sharing

- How keep nuclear code secret in British Army?

Secret Sharing

- How keep nuclear code secret in British Army?
- Burn it, but do not preserve integrity

How to Share a Secret Code I



1234567



How to Share a Secret Code I

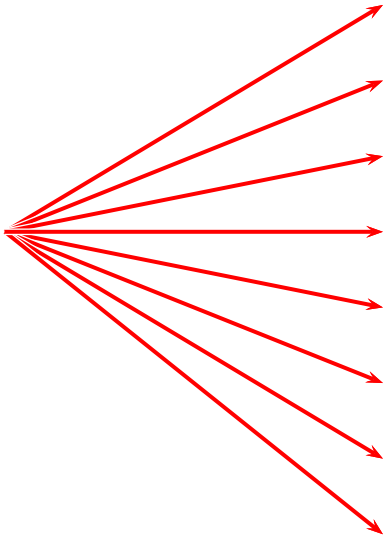


1234567



Problem of Integrity and Confidentiality

How to Share a Secret Code II



1234567



1234567



1234567

1234567



1234567

1234567



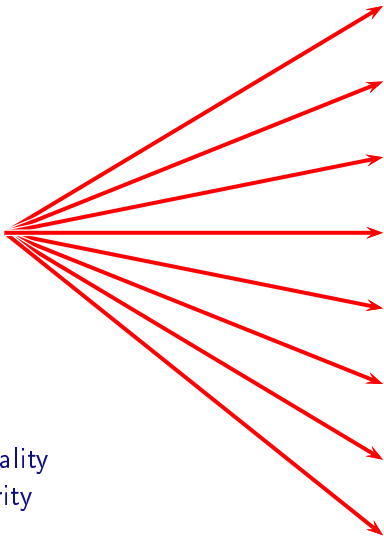
1234567



1234567

8779

How to Share a Secret Code II



1234567



1234567



1234567

1234567



1234567



1234567

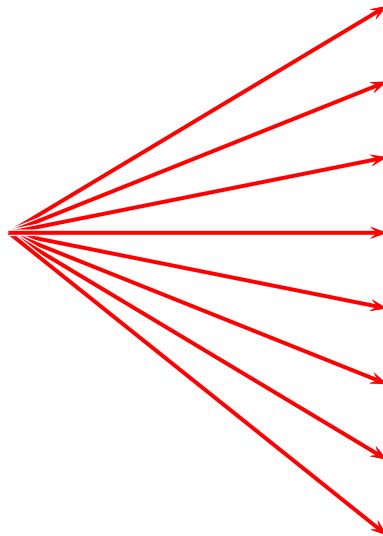
1234567



1234567

Problem of Confidentiality
No problem of Integrity

How to Share a Secret Code II



23572



11567



734567

534567



934567



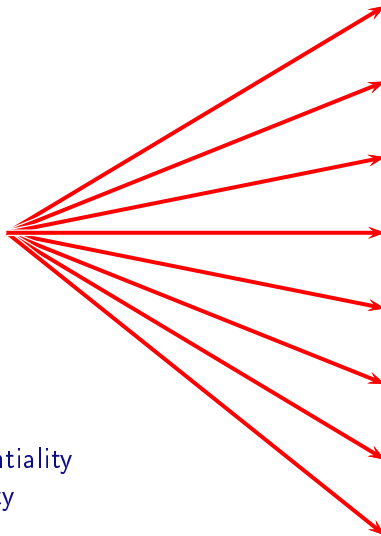
563317

114567



455967⁹

How to Share a Secret Code II



23572



11567



734567

534567



934567



563317

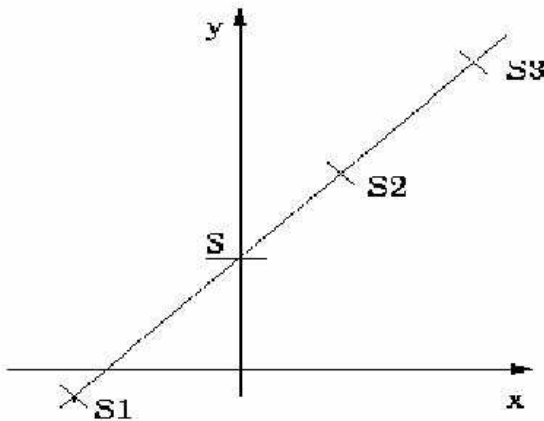
114567



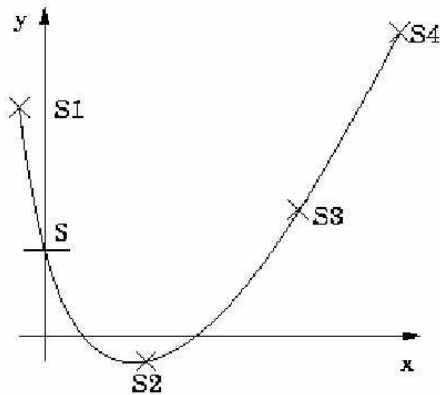
455867⁹

No Problem of Confidentiality
Problem of Integrity

(2,5)



(3,5)



(m, n)

It takes $n + 1$, points to define a polynomial of degree n .
Using Lagrange interpolation to recover the secret

Lagrange Interpolation

The Lagrange interpolating polynomial is the polynomial $L(x)$ of degree $\leq (n - 1)$ that passes through the n points $(x_1, y_1 = f(x_1)), (x_2, y_2 = f(x_2)), \dots, (x_n, y_n = f(x_n))$, and is

$$L(x) := \sum_{j=0}^k y_j \ell_j(x)$$

$$\ell_j(x) := \prod_{i=0, i \neq j}^k \frac{x - x_i}{x_j - x_i} = \frac{(x - x_0) \dots (x - x_{j-1}) (x - x_{j+1}) \dots (x - x_k)}{(x_j - x_0) \dots (x_j - x_{j-1}) (x_j - x_{j+1}) \dots (x_j - x_k)}$$

Shamir 1979

Initialization

Dealer chooses his secret $k \in Z_p$

Distribution

Dealer generates

$$P(X) = k + \sum_{i=1}^{m-1} a_i X^i$$

Send $s_i = (i, P(i))$ to each participant i

Reconstruction

Using Lagrange interpolation, with m distinct parts we compute $P(x)$

Weakness of Shamir

- In Shamir Secret Sharing there is no mechanism to identify if a share is valid or not.
- A total confidence is done to the dealer.

Weakness of Shamir

- In Shamir Secret Sharing there is no mechanism to identify if a share is valid or not.
- A total confidence is done to the dealer.

Verifiable Secret Sharing introduced by Feldman 1987, based on Discret Logarithm.

Verifiable Secret Sharing

Distribution like Shamir

Dealer generates

$$P(X) = k + \sum_{i=1}^{m-1} a_i X^i$$

Send $s_i = (i, P(i))$ to each participant i

Plus

Each server received : $g^k, g^{a_1}, \dots, g^{a_{m-1}}$

Verification

$$g^k \prod_{j=1}^{m-1} ((g^{a_j})^i)^j = g^{s_i}$$

Outline

- 1 Link between Computational and Symbolic
- 2 Crypto Verif
- 3 E-Cash
- 4 Funny Introduction to Zero Knowledge Proof
 - Principle
 - Funny example: Rubik's Cube
 - Example: Graph Coloring
- 5 Secret Sharing
- 6 Conclusion

Summary

Today

- Link Symbolic and Computational
- Pattern
- Key cycles
- Crypto Verif
- E-Cash
- Zero-Knowledge
- Secret Sharing

Next Time

- Exam, all lectures
- All “paper” documents are authorized.

1h30 PL + 1h30 JLR = 3h00

Friday December 12th 2008

9h00 - 12h00

Room D208

Thank you for your attention



Questions ?