

# Advanced Cryptography

## 1st Semester 2008-2009

### Non Interference and others

### Lecture 11

Pascal Lafourcade

*Université Joseph Fourier, Verimag*

Master: November 12th 2009 thanks to Bruno Blanchet, Bruce Kapron and Steve Kremer

## Last Time (I)

### Lecture

- ▶ Access Control
- ▶ Side Channel Attacks

# Outline of Today

## Non-Interference

- Motivation

- Approach using Type System

- Jif

## Secret Sharing

## Funny Introduction to Zero Knowledge Proof

- Principle

- Funny example: Rubik's Cube

- Example: Graph Coloring

## Conclusion

# Outline

## Non-Interference

- Motivation

- Approach using Type System

- Jif

## Secret Sharing

## Funny Introduction to Zero Knowledge Proof

- Principle

- Funny example: Rubik's Cube

- Example: Graph Coloring

## Conclusion

# Motivation

Malicious and/or buggy code is a threat:

- ▶ code from untrusted source (applet, javascript, ...)
- ▶ code interacts with your confidential data and with the outside world

Problem: does program  $p$  leak information?

Information flow analysis: Cohen, Denning in the 70's.

Non-interference (Goguen & Meseger) : semantic definition of absence of information leakage.

# Non-interference

The basic scenario:

- ▶ Public variables  $\ell, \ell_1, \dots$
- ▶ Secret variables  $h, h_1, h_2, \dots$

## Explicit leak

Program  $\ell := h_1$  leaks: the value of  $h$  is copied into  $\ell$ .

# Non-interference

The basic scenario:

- ▶ Public variables  $\ell, \ell_1, \dots$
- ▶ Secret variables  $h, h_1, h_2, \dots$

## Explicit leak

Program  $\ell := h_1$  leaks: the value of  $h$  is copied into  $\ell$ .

## Implicit leak

Program **if**  $h_1 = h_2$  **then**  $\ell := 0$  **else**  $\ell := 1$  **fi** leaks: the final value of  $\ell$  depends on the value of  $h_1$  and  $h_2$ .

# Non-interference

## Non-interference

The formal definition of *Non-interference* depends on the type of the considered system:

- ▶ Deterministic.
- ▶ Non-deterministic.
- ▶ Probabilistic.
- ▶ Cryptographic.

## Non-interference definitions

Henceforth, let  $L$  (resp.  $H$ ) be a set of low (resp. high) variables.

Deterministic case.

The denotation of a program is a mapping  $f : \Sigma \rightarrow \Sigma$ . Then,  $f$  is *non-interfering*, if

$$\forall S, S' \in \Sigma \cdot S_L = S'_L \Rightarrow f(S)_L = f(S')_L$$

Non-deterministic case (Possibilistic non-interference).

The denotation of a program is a mapping  $f : \Sigma \rightarrow 2^\Sigma$ . Now,  $f$  is said *non-interfering*, if

$$\forall S, S' \in \Sigma \cdot S_L = S'_L \Rightarrow f(S)_L = f(S')_L, \text{ where}$$

$A_L = \{a_L \mid a \in A\}$ , i.e.  $\forall S_1 \in f(S) \exists S_2 \in f(S') \cdot S_{1L} = S_{2L}$  and vice versa.

## Non-interference definitions (cntd.)

### Probabilistic case.

Let  $\mathcal{D}(\Sigma)$  be the set of distributions on  $\Sigma$ . The denotation of a program is a mapping  $f : \mathcal{D}(\Sigma) \rightarrow \mathcal{D}(\Sigma)$ .

For a distribution  $D \in \mathcal{D}(\Sigma)$ , define  $D_L \in \mathcal{D}(\Sigma_L)$  with

$$D_L(S_L) = \sum_{S' \in \Sigma, S'_L = S_L} D(S').$$

$f$  is *non-interfering*, if

$$\forall D, D' \in \mathcal{D}(\Sigma) \cdot D_L = D'_L \Rightarrow f(D)_L = f(D')_L$$

## Example

Possibilistic NI does not imply probabilistic NI:

$$(l := (h1 = h2)) \oplus_{\frac{1}{2}} (l := 0 \oplus_{\frac{1}{2}} l := 1)$$

## Example

Possibilistic NI does not imply probabilistic NI:

$$(\ell := (h_1 = h_2)) \oplus_{\frac{1}{2}} (\ell := 0 \oplus_{\frac{1}{2}} \ell := 1)$$

### Possibilistic NI

- ▶  $h_1 = h_2$  then  $\ell = 0$  and  $\ell = 1$
- ▶  $h_1 \neq h_2$  then  $\ell = 0$  and  $\ell = 1$

## Example

Possibilistic NI does not imply probabilistic NI:

$$(\ell := (h_1 = h_2)) \oplus_{\frac{1}{2}} (\ell := 0 \oplus_{\frac{1}{2}} \ell := 1)$$

### Possibilistic NI

- ▶  $h_1 = h_2$  then  $\ell = 0$  and  $\ell = 1$
- ▶  $h_1 \neq h_2$  then  $\ell = 0$  and  $\ell = 1$

### No Probabilistic NI

- ▶  $h_1 = h_2$  then  $P[\ell = 0] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$  and  $P[\ell = 1] = \frac{1}{4}$
- ▶  $h_1 \neq h_2$  then  $P[\ell = 0] = \frac{1}{4}$  and  $P[\ell = 1] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$

## Cryptographic non-interference

The denotation of a program is a function  $f : \Sigma \rightarrow \mathcal{D}(\Sigma)$  which can be canonically lifted to a function  $f : \mathcal{D}(\Sigma) \rightarrow \mathcal{D}(\Sigma)$ .

1. if  $\mathcal{D}_L \sim \mathcal{D}'_L$  then  $f(\mathcal{D})_L \sim f(\mathcal{D}')_L$ .
2.  $[x \leftarrow \mathcal{D} : (f(x)_L, x_H)] \sim [x, y \leftarrow \mathcal{D} : (f(x)_L, y_H)]$  [Laud]
3.  $|\Pr[f(S[h := 1])(\ell) = 1] - \Pr[f(S[h := 0])(\ell) = 1]|$  is negligible [SA'06].
4. Semantic security.
5. Possibilistic-symbolic [AHS'06]:

A symbolic semantics à la Dolev-Yao combined with the Possibilistic non-interference notion:

$\ell := \text{Enc}(k, 0); \text{if } h \text{ then } \ell_1 := \text{Enc}(k, 0) \text{ else } \ell_1 := \ell \text{ fi}$

$\Rightarrow$  distinguish encryption occurrences.

## Problem statement

Our aim is to automatically check cryptographic non-interference of programs that use random assignments and deterministic encryption: if  $\mathcal{D}_L \sim \mathcal{D}'_L$  then  $f(\mathcal{D})_L \sim f(\mathcal{D}')_L$

### Methods to ensure Non-Interference

- ▶ At Compile time:
  - ▶ Static Analysis
  - ▶ Type Checking (Volpano & Smith)
- ▶ At run-time : monitoring

# Example

Simple programs

```
T[3] = h; l = T[3];
```

# Example

## Simple programs

```
T[3] = h; l = T[3];
```

leaks

## Example

### Simple programs

```
T[3] = h; l = T[3];  
leaks T[3] = l; h = T[3];  
Does not leak
```

```
for (i=0; i<T.length; i++) T[i]=0;  
T[h]=1;
```

## Example

### Simple programs

```
T[3] = h; l = T[3];  
leaks T[3] = l; h = T[3];  
Does not leak
```

```
for (i=0; i<T.length; i++) T[i]=0;  
T[h]=1;  
leaks information
```

## Motivating Example (cf. [Volpano'00])

Encryption does not guarantee absence of leakage *per se*:

```

 $l := 0^\eta; m := 0^{\eta-1}1;$ 
for  $i := \eta$  to 1 do    $\ell_1 := \text{Enc}(k, h|m);$ 
                            $\ell_2 := \text{Enc}(k, h);$ 
                           if  $(\ell_1 = \ell_2)$  then  $l := l|m$  else skip fi ;
                            $m := m \ll 1$  od

```

( $m, \ell, \ell_1, \ell_2$  low-security variables,  $k, h$  high-security variables,  
parameter  $\eta$  size of blocks)

## Example-cntd.

```

 $l := 0^\eta; m := 0^{\eta-1}1;$ 
for  $i := \eta$  to 1 do
     $l_1 := \text{Enc}(k, \nu l_r \cdot (h|m) + l_r);$ 
     $l_2 := \text{Enc}(k, h + l_r);$ 
    if  $(l_1 = l_2)$  then  $l := l|m$  else skip fi ;
   $m := m \ll 1$  od

```

# Example cntd

```

 $l := 0^\eta; m := 0^{\eta-1}1;$ 
for  $i := \eta$  to 1 do
     $l_1 := \text{Enc}(k, \nu l_r \cdot (h|m) + l_r);$ 
     $l_2 := \text{Enc}(k, \nu l_r \cdot h + l_r);$ 
    if  $(l_1 = l_2)$  then  $l := l|m$  else skip fi ;
   $m := m \ll 1$  od

```

# Objectives

- ▶ Develop a type system for non-interference for programs that use deterministic encryption
- ▶ Establish soundness in the exact (concrete) security model

Computationally Sound Typing for Non-interference: The Case of Deterministic Encryption, by Judicaël Courant and Cristian Ene and Yassine Lakhnech, at FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007.

# A type system for a simple imperative language

Expressions:

$$\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau}$$

Sub-typing:  $L \sqsubseteq H$

$$\frac{\Gamma \vdash S : \tau' \quad \tau \sqsubseteq \tau'}{\Gamma \vdash S : \tau}$$

Commands:

$$\frac{(x, \tau) \in \Gamma \quad \Gamma \vdash e : \tau}{\Gamma \vdash x := e}$$

$$\frac{\Gamma \vdash S_1 : \tau \quad \Gamma \vdash S_2 : \tau}{\Gamma \vdash S_1; S_2 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash f(e_1, e_2) : \tau}$$

$$\frac{\Gamma \vdash e : \tau \quad \tau \sqsubseteq \tau'}{\Gamma \vdash e : \tau'}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash S_1 : \tau \quad \Gamma \vdash S_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 \text{ fi} : \tau}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash S : \tau}{\Gamma \vdash \text{while } e \text{ do } S \text{ od} : \tau}$$

## Approach using Type System

Using the same type system but something more is needed.

Sound but what about  $\ell_1 := h_1 \oplus \text{random}()$ ?

- ▶ Ill-typed for usual type systems.
- ▶ Does not leak information (equivalent to  $\ell_1 := \text{random}()$ ).
- ▶ We want to accept such a program (one-time pad).

# Quizz

Should we typecheck/What kind of type should we introduce to typecheck

- ▶  $l_1 := \nu h_2 \cdot h_2 \oplus h_1?$
- ▶  $l_1 := \nu l_2 \cdot l_2 \oplus h_1?$
- ▶  $l_1 := \pi(h_1)?$  ( $\pi$  random permutation)
- ▶  $l_1 := \pi(\nu l_2 \cdot l_2 \oplus h_1)?$

Types:  $(\tau, \theta)$  where

- ▶  $\tau \in \{L, H\}$
- ▶  $\theta \in \{L^r, H^r, \top\}$

## Type systems for non-interference

- ▶ Introduce type  $L$  for low-sensitive data/variables and  $H$  high-sensitive data/variables.
- ▶ Arithmetic operator/assignment apply to and return data of the same type.
- ▶ Subtyping: expressions in  $L$  also belong to  $H$ .

Examples:

$$\begin{array}{l}
 h_1 := h_2 \\
 h_1 := \ell_1 \\
 h_1 := \ell_1 \oplus h_2 \\
 \ell_1 := h_1 \oplus h_2 \quad ?
 \end{array}
 \quad
 \begin{array}{l}
 \frac{(x, \tau) \in \Gamma \quad \Gamma \vdash e : \tau}{\Gamma \vdash x := e : \tau} \\
 \frac{\Gamma \vdash e : \tau \quad \tau \sqsubseteq \tau'}{\Gamma \vdash e : \tau'} \quad L \sqsubseteq H \\
 \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \oplus e_2 : \tau}
 \end{array}$$

## Non-interference with random permutation

Do these programs leak:

- ▶  $l_1 := \pi(h_1 \oplus \nu l_2 \cdot l_2)$
- ▶  $l_1 := \pi(h_1 \oplus \nu l_2 \cdot l_2); l_3 := \pi(l_2)$

Problem: once used to randomize argument of a permutation,  $l_2$  must not appear under a permutation.

→ We need to introduce a set of forbidden variables (not reuse random variable in  $\pi$ ), and add this set to the premises of our typing judgments.

## An example CBC

$$\begin{aligned} & \nu l_0; \\ & l_1 := \text{Enc}(k, l_0 \oplus h_1); \dots ; l_i := \text{Enc}(k, l_{i-1} \oplus h_i), \dots, \\ & l_n := \text{Enc}(k, l_{n-1} \oplus h_n) \end{aligned}$$

- ▶  $l_0$  is typed  $(L, L')$ .
- ▶  $l_i \oplus h_{i+1}$  is typed  $(H, L')$ .
- ▶  $\text{Enc}(k, l_i \oplus h_i)$  is typed  $(L, L')$ .
- ▶  $l_i$  is added to  $G$ .

# Type system for non-interference of randomized programs

New typing rules for expressions:

$$\frac{\Gamma \vdash e_1 : (\tau_1, \theta_1) \quad \Gamma \vdash e_2 : (\tau_2, \theta_2)}{\Gamma \vdash e_1 \oplus e_2 : (\tau_1 \sqcup \tau_2, \theta_1 \sqcap \theta_2)}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash \nu x \cdot x : (\tau, \tau')}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : (\tau, \top)}$$

For programs without random permutation

Soundness result: well-typedness implies non-interference.

# Proof of non-interference for random permutation

Result: soundness of typing for random permutation.

## The result

For any type environment  $\Gamma$ , command  $c$  and pairs of distribution ensembles in  $\mathcal{D}(\Sigma)$  such that

- ▶  $\Gamma, \emptyset, \emptyset \vdash c : \tau$ , for some type  $\tau \in \{H, L\}$ , and
- ▶  $X|_L \sim_{(t, \epsilon)} Y|_L$ ,

we have

$$\llbracket c \rrbracket(X)|_L \sim_{(t - \mathbb{T}(c), \epsilon')} \llbracket c \rrbracket(Y)|_L \text{ where } \epsilon' = \frac{2\mathcal{T}(c)^2}{|\mathcal{U}|} + \epsilon.$$

- ▶  $\mathcal{T}(c)$  the number of calls of  $\pi$
- ▶  $\mathbb{T}(c)$  the running time of  $c$

## Main ideas of the proof

Ideas:

- ▶ Reduction to programs without permutation
- ▶ Distinguishing occurrences of permutation applied to random expressions ( $\pi^r$ ) and to low security expressions ( $\pi^\top$ )
- ▶ Simulate  $\pi^\top$  by an unknown function and  $\pi^r(e)$  by random sampling (independent of its argument).

Simulation might fail, but with quantifiable low risk.

$$\llbracket c^P \rrbracket(X)_{|L} \sim_{sim} \llbracket c \rrbracket(X)_{|L} \sim_{1st} \llbracket c \rrbracket(Y)_{|L} \sim_{sim} \llbracket c^P \rrbracket(Y)_{|L}$$

## Non-interference with encryption

Now, we replace the permutation  $\pi$  by encryption.

### Cipher

A cipher block is a *family of permutations*  $\Pi : \mathbf{Keys}(\Pi) \times \mathcal{D} \rightarrow \mathcal{D}$ , where  $\mathbf{Keys}(\Pi)$  is the key space of  $\Pi$ , and for any  $k \in \mathbf{Keys}(\Pi)$ ,  $\Pi(k, \cdot)$  is a permutation onto  $\mathcal{U}$ .

## Security assumption

The usual security notion for ciphers is Pseudo-randomness.

$$\begin{aligned}
 &\text{Experiment } \mathbf{PRP}_b^\eta(\mathcal{A}) : \\
 &k \stackrel{r}{\leftarrow} \mathbf{Keys}(\Pi); \mathcal{P} \stackrel{r}{\leftarrow} \mathcal{SP}; \\
 &\mathcal{O}_0 = \mathcal{P}; \mathcal{O}_1 = \text{Enc}(k, \cdot); \\
 &b' \leftarrow \mathcal{A}^{\mathcal{O}_b}()
 \end{aligned}$$

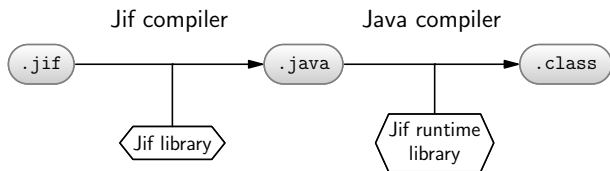
The **PRP** advantage of  $\mathcal{A}$  is defined as

$$\text{Adv}_{\Pi}^{\text{PRP}}(\mathcal{A}) = \Pr[\mathbf{PRP}_1^\eta(\mathcal{A}) = 1] - \Pr[\mathbf{PRP}_0^\eta(\mathcal{A}) = 1].$$

An encryption scheme  $\Pi$  is said to be a  $(t, \epsilon)$ -pseudo-random permutation, denoted  $(t, \epsilon)$ -**PRP**, if for any PRP adversary  $\mathcal{A}$ , that runs in time  $t$ ,  $\text{Adv}_{\Pi}^{\text{PRP}}(\mathcal{A}) \leq \epsilon$ .

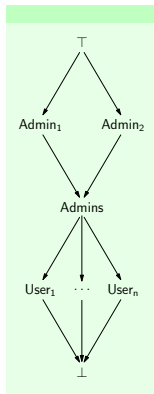
# Jif (Java + information flow)

- ▶ Jif is an extension of the Java language by Andrew C. Myers.
  - ▶ for confidentiality and integrity enforcement
  - ▶ static IF analysis with possible runtime checks



- ▶ Jif relies on a sparse security policy in the code
  - ▶ defined by *labels*
  - ▶ build upon *principals*

## Jif principals



- ▶ A principal is a role in the system
- ▶ Hierarchical (lattice) organisation of principals
  - ▶ a principal can *act for* another ( $\rightarrow$ )
- ▶ Two constant principals
  - ▶ top ( $\top$  or  $*$ ) can act for every principal
  - ▶ every principal can act for bottom ( $\perp$  or  $\_$ )
- ▶ User defined principal = a class implementing the `jif.lang.Principal` interface

## Jif labels I

- ▶ A label describes dual aspects of a security policy
  - ▶ confidentiality  $policyOwner \rightarrow readerAllowed$
  - ▶ integrity rule  $policyOwner \leftarrow writerAllowed$

### Example

```
int {X->X,Y; X<-X,Y\} a;
int {X->X; X<-X} b;
```

```
a = b;           / confidentiality problem
b = a;           / integrity problem
```

## Confidentiality policy

*policyOwner*  $\rightarrow$  *readerAllowed*

### Conjunction (; or $\sqcap$ )

$c \sqcap d$  permits a principal to read iff both  $c$  and  $d$  allow it.

### Disjunction (meet or $\sqcap$ )

$c \sqcap d$  permits a principal to read iff either  $c$  or  $d$  allows it.

### Ordering relation $\sqsubseteq_C$

“ $c$  is no more restrictive than  $d$ ” ( $c \sqsubseteq_C d$ ) iff for every owner  $o$  the set of readers allowed by  $c$  is included in the set of readers included by  $d$ .

## Integrity policy

$$policyOwner \leftarrow writerAllowed$$

### Disjunction (; or $\sqcup$ )

$c \sqcup d$  permits a principal to write if either  $c$  or  $d$  allows it.

### Conjunction (meet or $\sqcap$ )

$c \sqcap d$  permits a principal to write iff both  $c$  and  $d$  allow it.

### Ordering relation $\sqsubseteq_I$

“ $c$  is no more restrictive than  $d$ ” ( $c \sqsubseteq_I d$ ) iff for every owner  $o$  the set of writers allowed by  $c$  is included in the set of writers included by  $d$ .

# Outline

## Non-Interference

- Motivation

- Approach using Type System

- Jif

## Secret Sharing

- Funny Introduction to Zero Knowledge Proof

  - Principle

  - Funny example: Rubik's Cube

  - Example: Graph Coloring

## 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

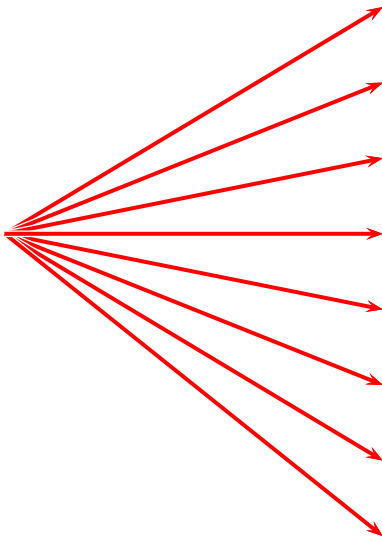


1234567



Problem of Integrity and Confidentiality

## How to Share a Secret Code II



1234567



1234567



1234567

1234567



1234567

1234567



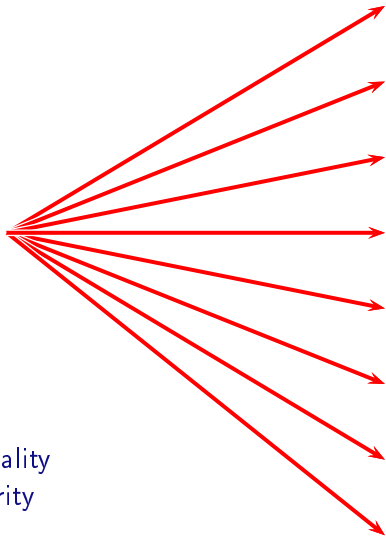
1234567



1234567

10/6/09

## How to Share a Secret Code II



1234567



1234567



1234567

1234567



1234567



1234567

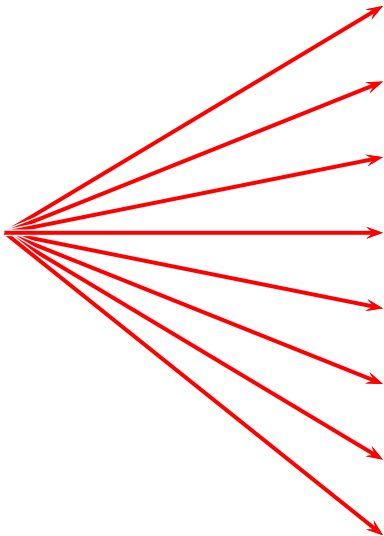
1234567



1234567

Problem of Confidentiality  
No problem of Integrity

## How to Share a Secret Code II



23572



11567



734567

534567



934567



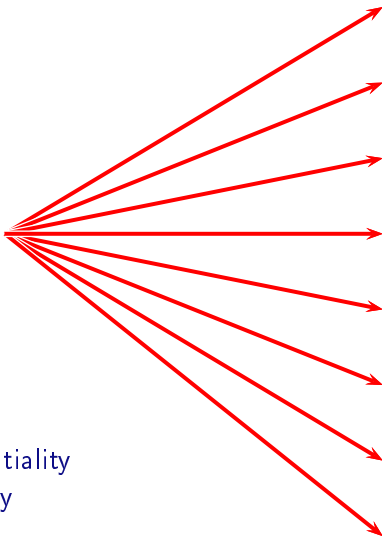
563317

114567



45567<sup>1</sup>

## How to Share a Secret Code II



23572



11567



734567

534567



934567



563317

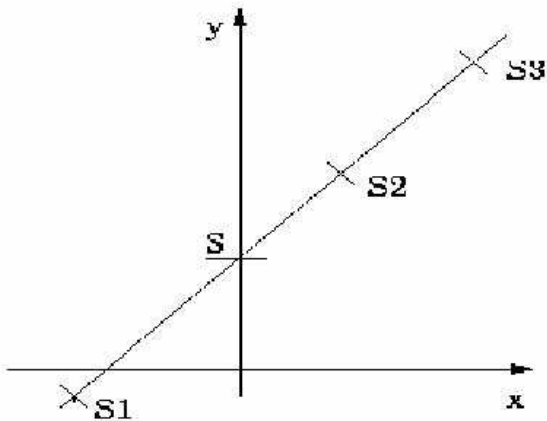
114567



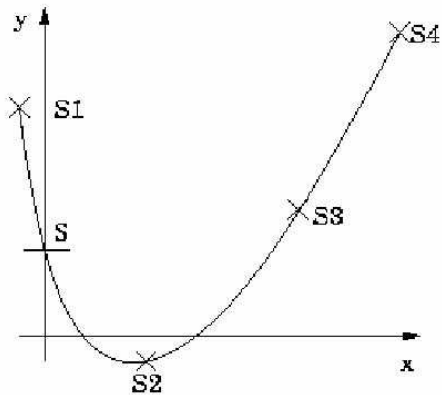
45567<sup>1</sup>

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  $k \in \mathbb{Z}_p$

## 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 Discrete 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

## Non-Interference

- Motivation

- Approach using Type System

- Jif

## Secret Sharing

## Funny Introduction to Zero Knowledge Proof

- Principle

- Funny example: Rubik's Cube

- Example: Graph Coloring

## Conclusion

# Interactive Zero Knowledge Proofs

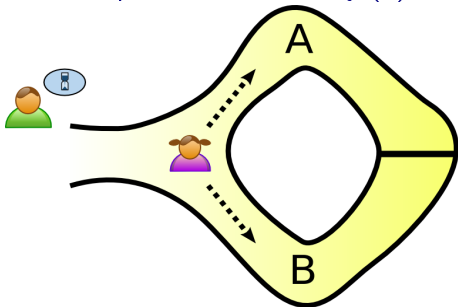
In an interactive zero knowledge proof, a prover  $P$  interacts with a verifier  $V$  to demonstrate the validity of an assertion without revealing anything about the assertion to the verifier.

## 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

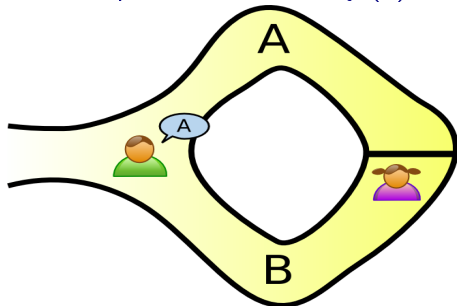
An Example: The Cave Story (2)



First, Victor waits outside while Peggy chooses a path.

# Interactive Zero Knowledge Proofs

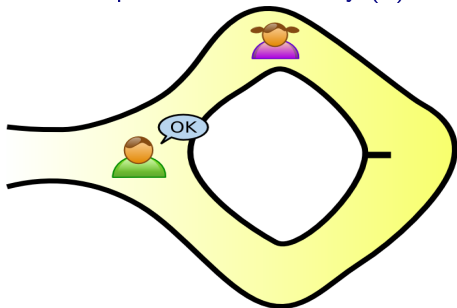
An Example: The Cave Story (3)



Then Victor enters and shouts the name of a path.

# Interactive Zero Knowledge Proofs

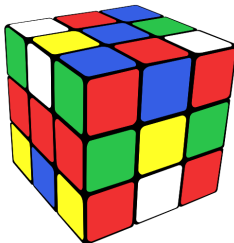
## An Example: The Cave Story (4)



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.

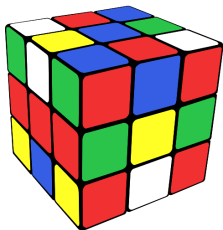


Question ?

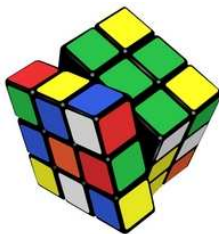
She wants to prove to Bob she has the skill to solved a given scrambled Rubik's cube without revealing it. How can she do it?

# Rubik's Cube

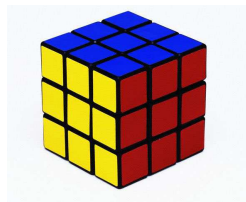
Alice scrambles the cube and proposed a new cube to Bob



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  $\pi$  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  $r_u, r_v, \pi(c(u)), \pi(c(v))$  which allows Bob to confirm messages send by Alice.

Given a permutation  $\pi$

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

$B \rightarrow A : u, v$

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

## 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

## Non-Interference

- Motivation

- Approach using Type System

- Jif

## Secret Sharing

## Funny Introduction to Zero Knowledge Proof

- Principle

- Funny example: Rubik's Cube

- Example: Graph Coloring

## Conclusion

# Summary

## Today

- ▶ Non-Interference
- ▶ Secret Sharing
- ▶ Zero-Knowledge

Thank you for your attention



Questions ?