Java sécurité

Sécurité bas-niveau

Quelques éléments de Sécurité bas-niveau

- Au niveau du code
  - les OS assurent certaines protections pour la sécurité bas niveau
  - anti-virus
  - pare-feux
  - sandbox Java

Sécurité bas-niveau

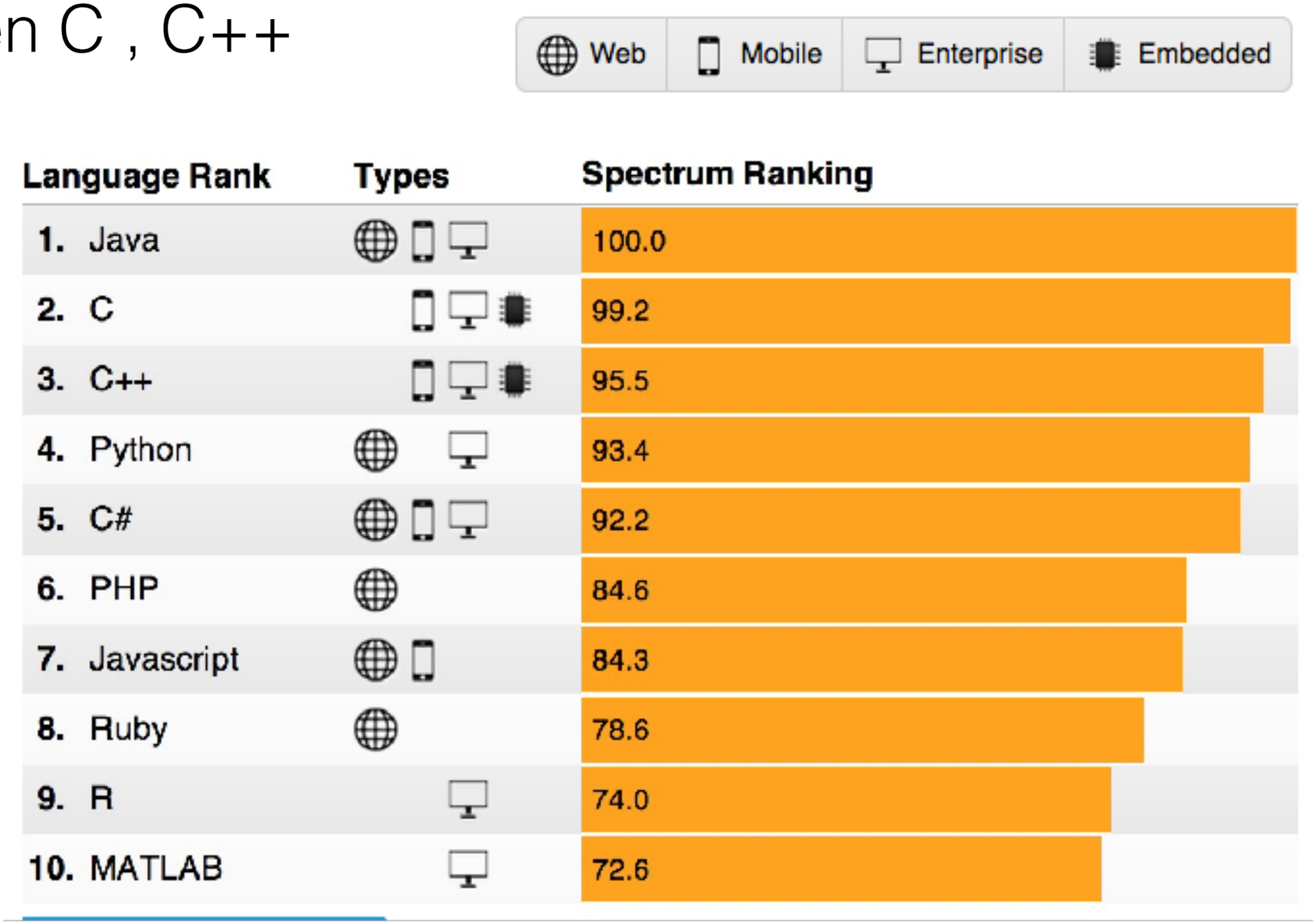
- Sécurité bas-niveau:
  - « bugs » des programmes
  - éviter les bugs par des bonnes pratiques de conceptions et d'implémentations
  - Exemple: un « warning » au cours d'une compilation correspond à une faille potentielle de sécurité

Buffer overflow

- Buffer overflow (accès à une zone non-allouée)
- bug (du programme) classique en C C++provoque un crash du programme mais permet à un attaquant de
  - voler des informations privées
  - corrompre des données
  - exécuter un code malveillant

Buffer overflow

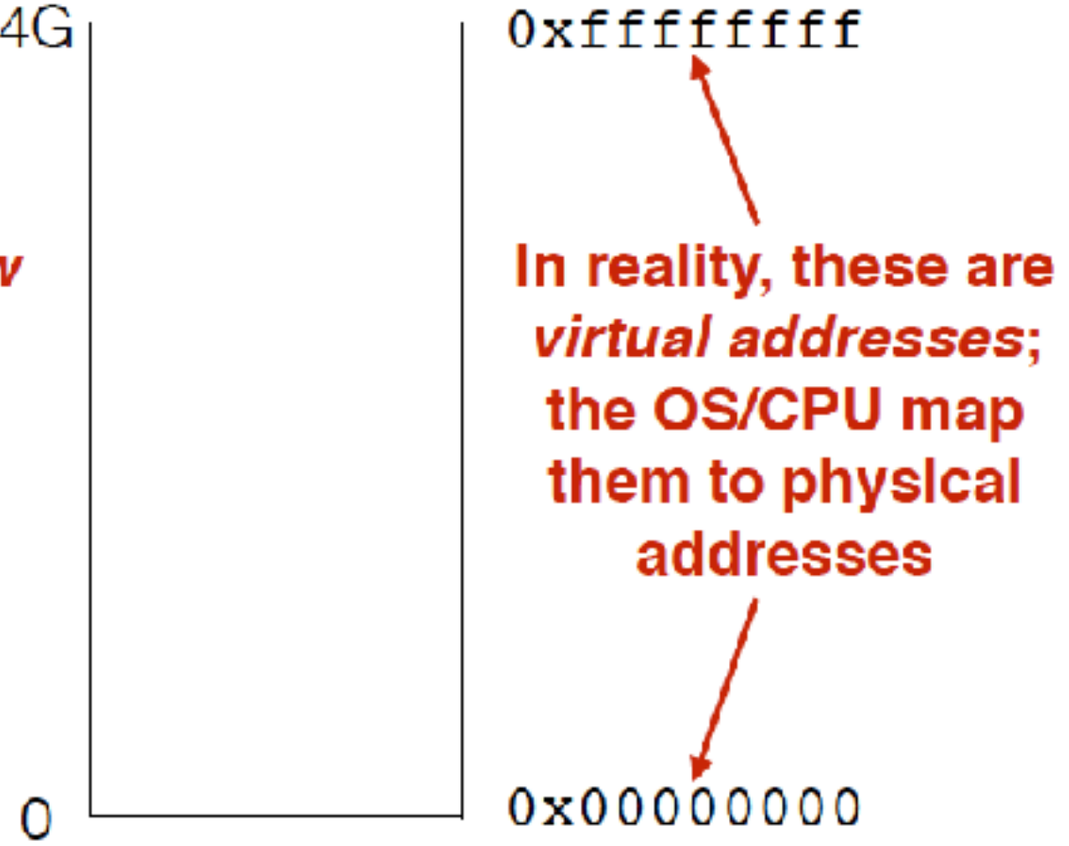
- un classique en C , C++
- pour mémoire:



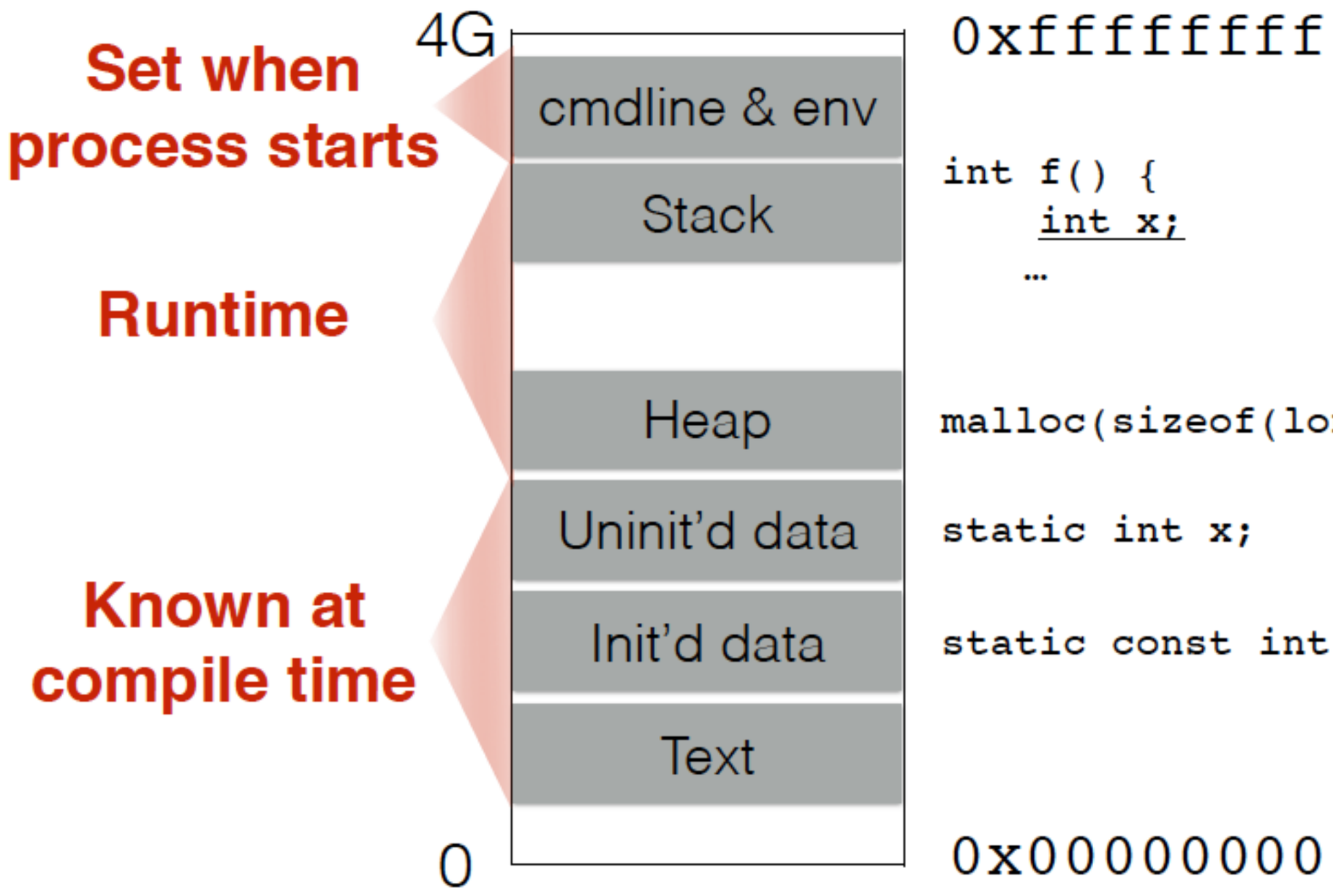
http://spectrum.ieee.org/static/interactive-the-top-programming-languages

pour mémoire

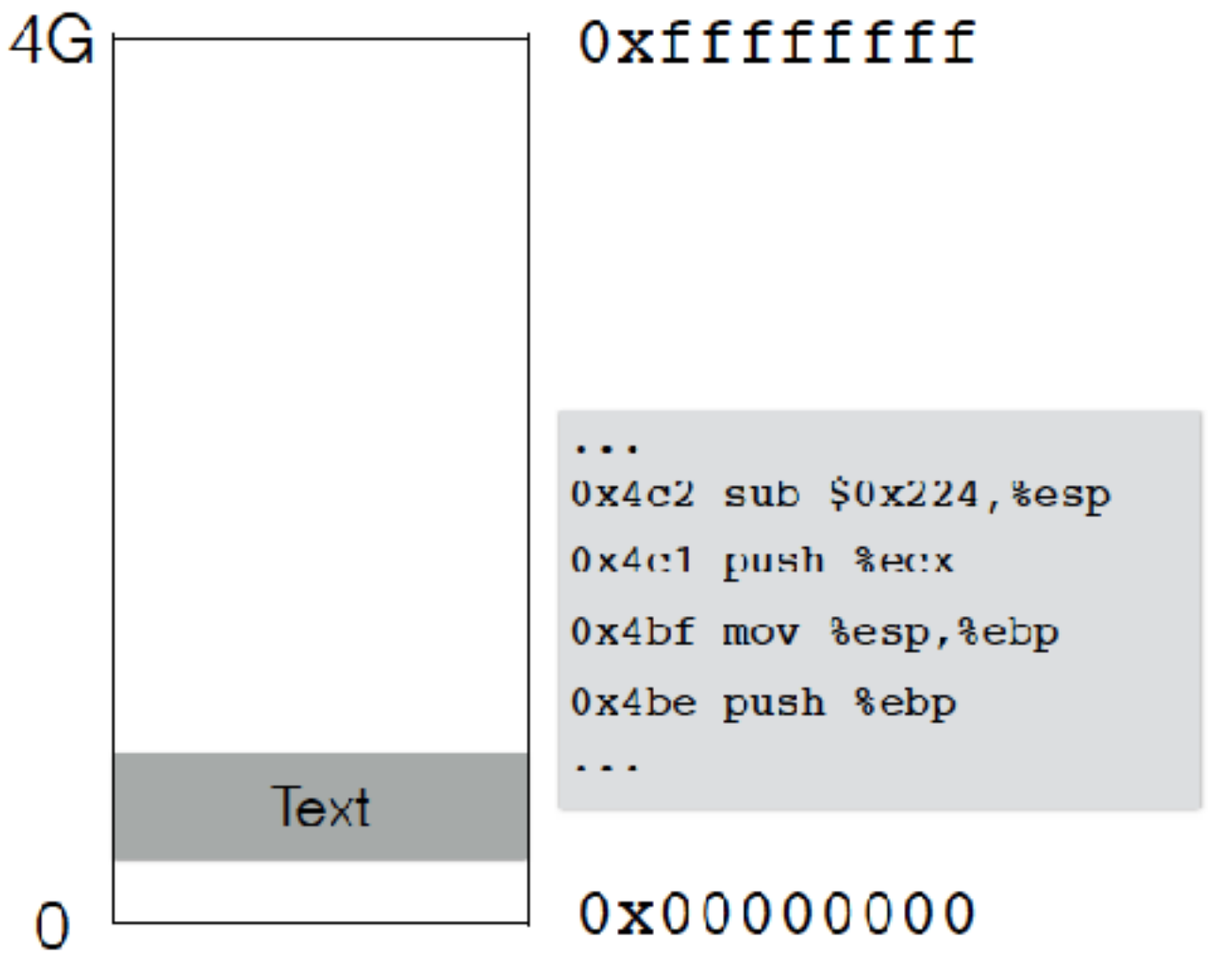
un programme en mémoire



Pour mémoire…

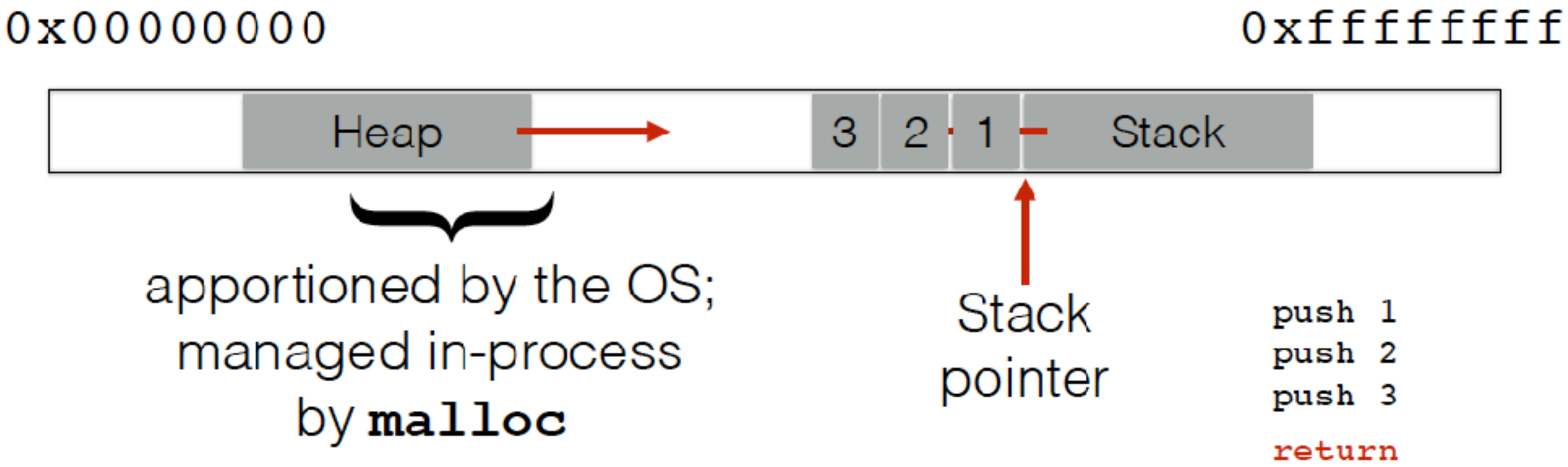


Pour mémoire

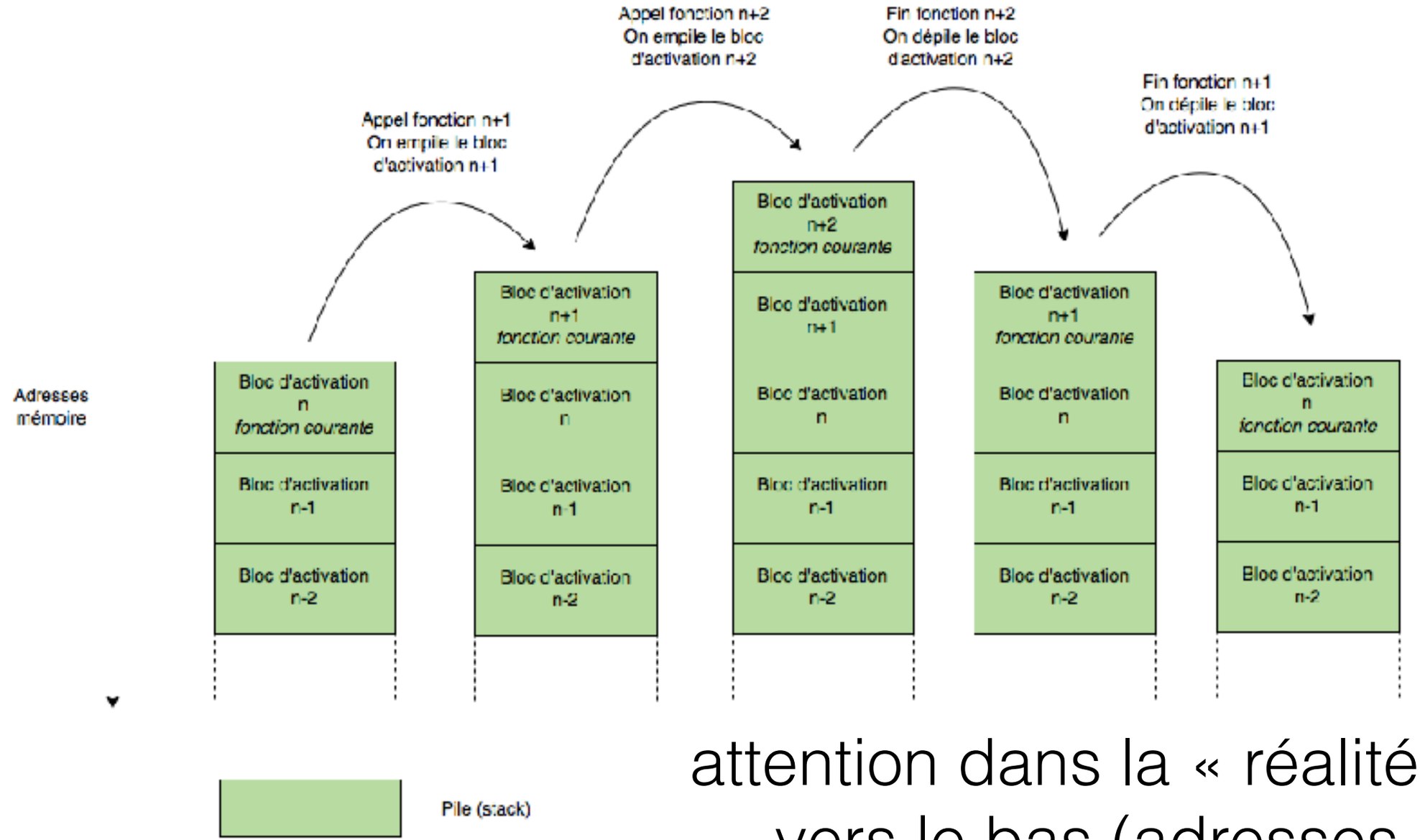


le code est dans la zone de texte

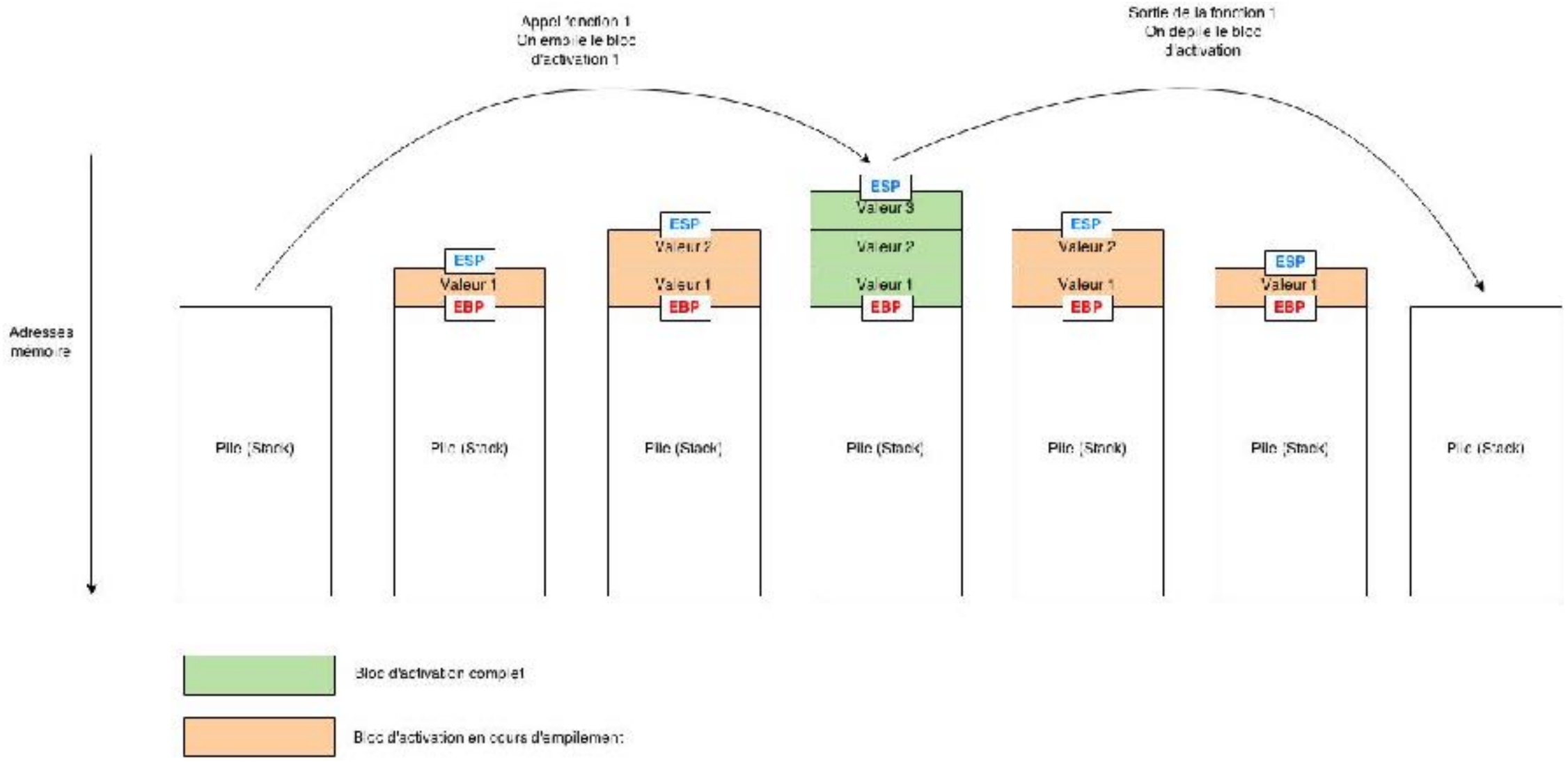
Allocation


0x00000000
0xffffffff
Heap
3
2
1
Stack
apportioned by the OS;
managed in-process
by malloc
Stack
pointer
push 1
push 2
push 3
return

Pile…



attention dans la « réalité »la pile va vers le bas (adresses basses)

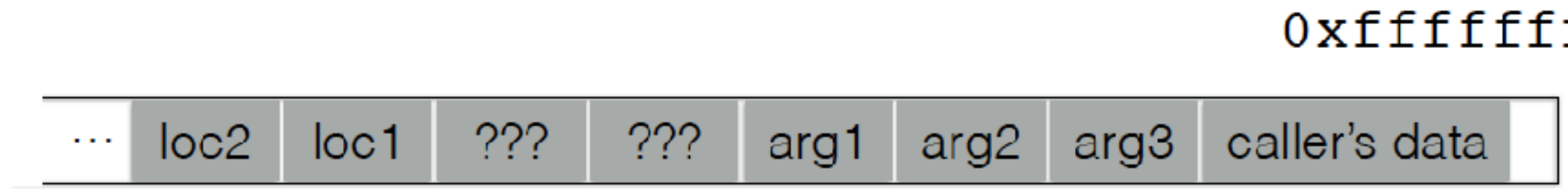


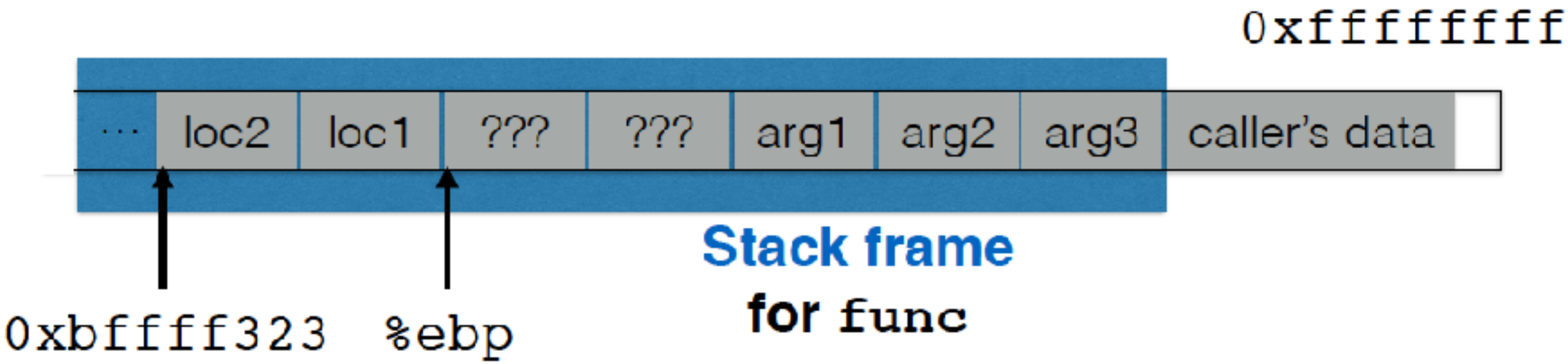
%ebp est l'adresse du début de la stack frame

%esp est l'adresse du haut de la pile

allocation sur la pile

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4]
    int  loc2;
    ...
}
```



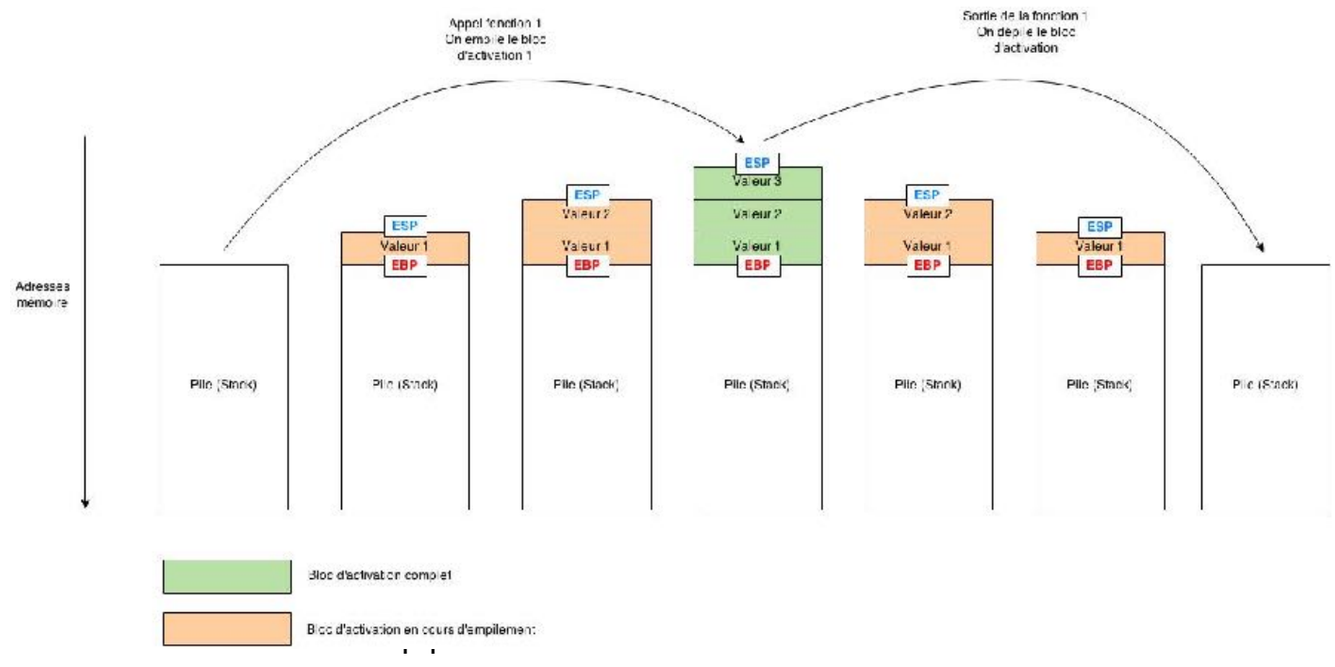


loc2 est à 8 byte avant %epb

%esp est l'adresse du haut de la pile

%ebp est l'adresse du début de la stack frame

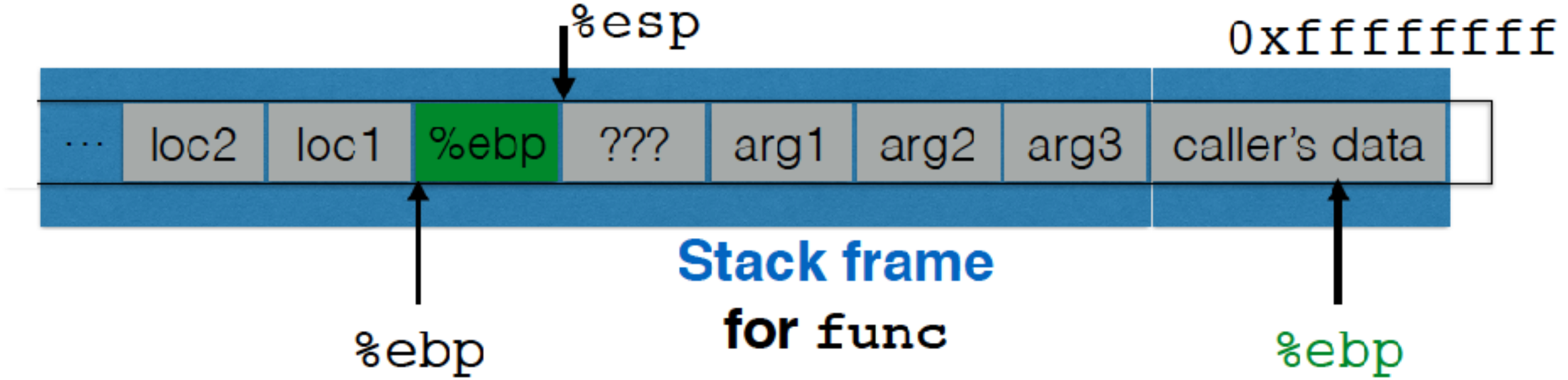
%eip est l'adresse de l'instruction courante



Sécurité

Pour le retour

```
int main()
{
     ...
     func("Hey", 10, -3);
     ...
}
```



push %ebp
set %ebp à la valeur courante (%esp)

set %epb to (%ebp) au retour

```c
#include <stdio.h>
#include <stdlib.h>

int reponse(int a, int b, int c) {
    return a+b+c;
}

int main() {
    int result;
    result = reponse(4, 8, 42);
}
```

```
Dump of assembler code for function main:
   0x080483a5 <+0>:     push   ebp
   0x080483a6 <+1>:     mov    ebp,esp
   0x080483a8 <+3>:     sub    esp,0x1c
   0x080483ab <+6>:     mov    DWORD PTR [esp+0x8],0x2a
   0x080483b3 <+14>:    mov    DWORD PTR [esp+0x4],0x8
   0x080483bb <+22>:    mov    DWORD PTR [esp],0x4
   0x080483c2 <+29>:    call   0x8048394
   0x080483c7 <+34>:    mov    DWORD PTR [ebp-0x4],eax
   0x080483ca <+37>:    leave
   0x080483cb <+38>:    ret
End of assembler dump.
```

```
(gdb) disas reponse
Dump of assembler code for function reponse:
   0x08048394 <+0>:     push   ebp
   0x08048395 <+1>:     mov    ebp,esp
   0x08048397 <+3>:     mov    eax,DWORD PTR [ebp+0xc]
   0x0804839a <+6>:     mov    edx,DWORD PTR [ebp+0x8]
   0x0804839d <+9>:     lea    eax,[edx+eax*1]
   0x080483a0 <+12>:    add    eax,DWORD PTR [ebp+0x10]
   0x080483a3 <+15>:    pop    ebp
   0x080483a4 <+16>:    ret
End of assembler dump.
```

appel de fonction

- fonction appelante:
    1. push des arguments sur la pile (en ordre inversé)
    2. push l'adresse de retour
    3. aller à l'adresse de la fonction
- fonction appelée
    4. push du pointeur de l 'ancienne frame sur la pile (%epb)
    5. set le pointeur de frame (%ebp) à la valeur de la fin actuelle de pile
    6. push les variables locales sur la pile
- retour de la fonction
    7. reset la précédente stack frame %esp=%esb, %ebp=(%ebp)
    8. aller à l'adresse de retour: %eip=(%esp)

Buffer overflow:

- Buffer: zone contiguë de mémoire associée à une variable. Une String en C est un tableau de char terminé par ‘\0’

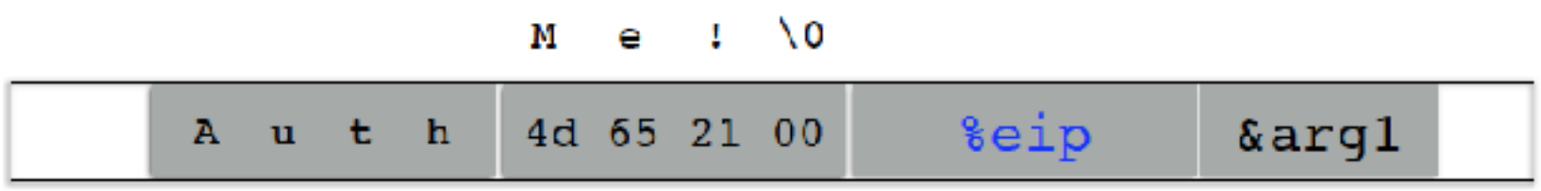
- Overflow: écrire dans le buffer plus que la zone réservée

- ce qui est écrit en trop « écrase » ou modifie une zone non allouée pour ça… un expert peut savoir laquelle et utiliser le buffer overflow de façon malveillante

Exemple

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}

int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

set %ebp to 0x0021654d

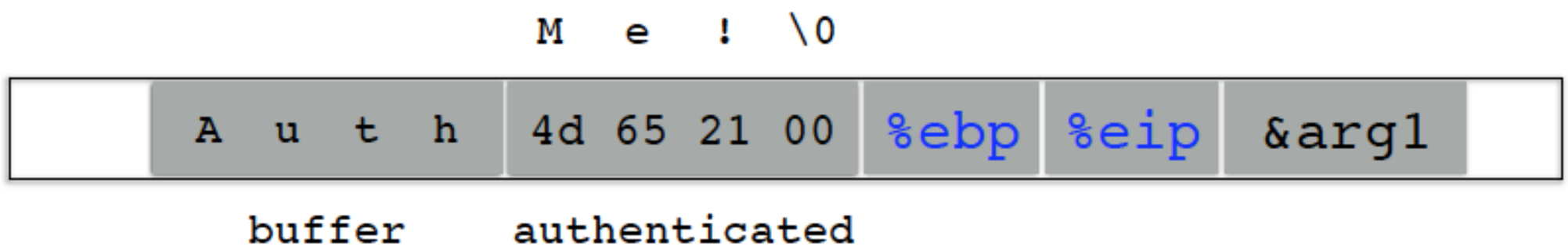


SEGFAULT (0x00216551)

Exemple

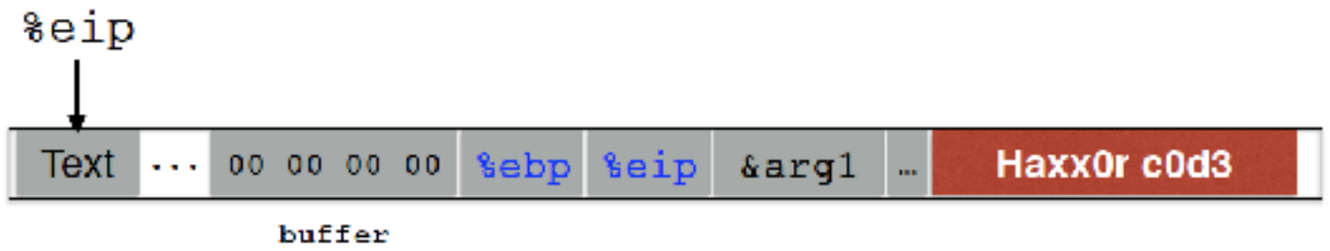
```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) { ...
}

int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

strcpy écrit jusqu'à '/0': on peut même de cette façon exécuter du code



injection de code

```
void func(char *arg1)
{
    char buffer[4];
    sprintf(buffer, arg1);
    ...
}
```



(1) Load my own code into memory

(2) Somehow get `%eip` to point to it

Injection de code…

- Mais il faut:
  - du code machine (prêt à être exécuté)
  - sans ‘\0’
  - sans utiliser le loader

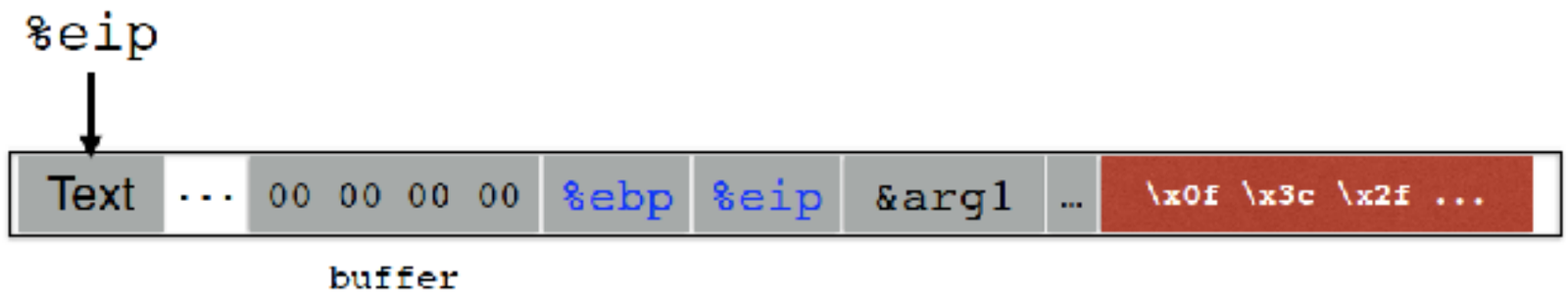
- un code permettant l’accès au système: exemple shell

```
#include <stdio.h>
int main( ) {
   char *name[2];
   name[0] = “/bin/sh”;
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

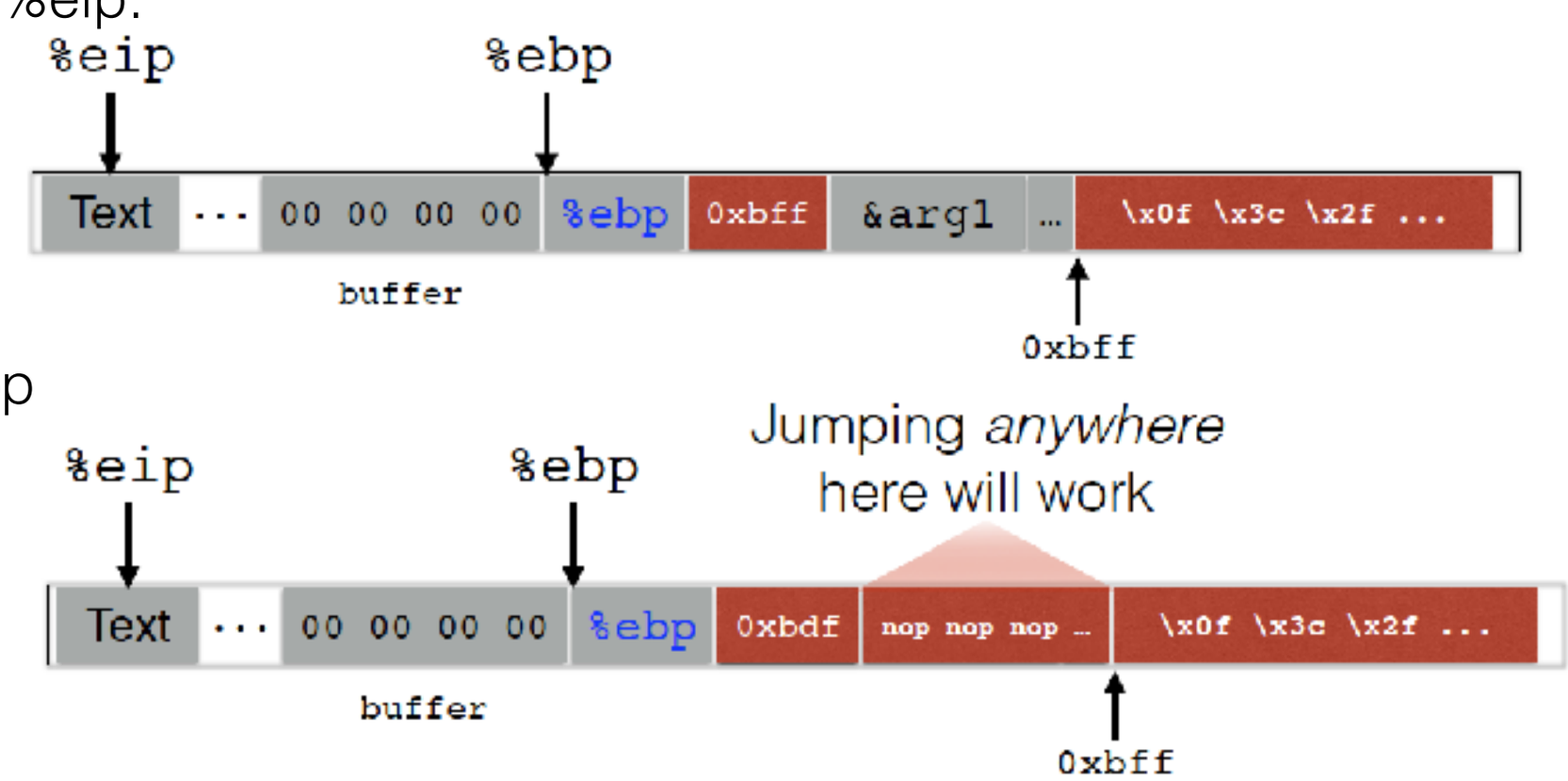
Assembly	Machine code
xorl %eax, %eax	“\x31\xc0”
pushl %eax	“\x50”
pushl $0x68732f2f	“\x68””//sh”
pushl $0x6e69622f	“\x68””/bin”
movl %esp,%ebx	“\x89\xe3”
pushl %eax	“\x50”
...	...

Injection de code

- En plus: faire un « jump » vers le code injecté, (où est-il?)



pirater le %eip.



des nop

mémoire

- Il existe de nombreuses autres attaques…
  - heap overflow: overflow d'un buffer alloué par malloc

```
typedef struct _vulnerable_struct {
  char buff[MAX_LEN];
  int (*cmp)(char*,char*);
} vulnerable;

int foo(vulnerable* s, char* one, char* two)
{
 strcpy( s->buff, one );     copy one into buff
 strcat( s->buff, two );     copy two into buff
 return s->cmp( s->buff, "file://foobar" );
}
```

si strlen(one) +strlen(two)>MAX_LEN
s->cmp est modifié!

Integer overflow

```
void vulnerable()
{
  char **response;
  int nresp = packet_get_int();
  if (nresp > 0) {
   response = malloc(nresp*sizeof(char*));
   for (i = 0; i < nresp; i++)
    response[i] = packet_get_string(NULL);
  }
```

HUGE

Wrap-around

Overflow

- If we set `nresp` to 1073741824 and `sizeof(char*)` is 4
- then `nresp*sizeof(char*)` overflows to become 0
- subsequent writes to allocated `response` overflow it

Dangling pointer

- Pointeur qui ne pointe pas vers le type d'objet approprié:
  - après un free, le pointeur est libéré mais peut continuer à être utilisé par le programme (bug)
  - la zone utilisée peut être modifiée.

Des solutions?

- stack smashing attacks:
  - interdire d'exécuter du code ne provenant pas de la zone de texte
  - charger les librairies de façon imprévisible
  - …
- changer (randomisation) les mouvements de la pile
- contrôler les flux: calculer à la compilation le graphe des appels, surveiller à l'exécution…
- Allocateur sûr
- typage sûr
- …

Surtout:

Secure coding

En java…

Java sécurité

- JVM: vérification du bytecode, vérifications à l'exécution (ex: dépassement des bornes des tableaux), garbage collector, sûreté du typage

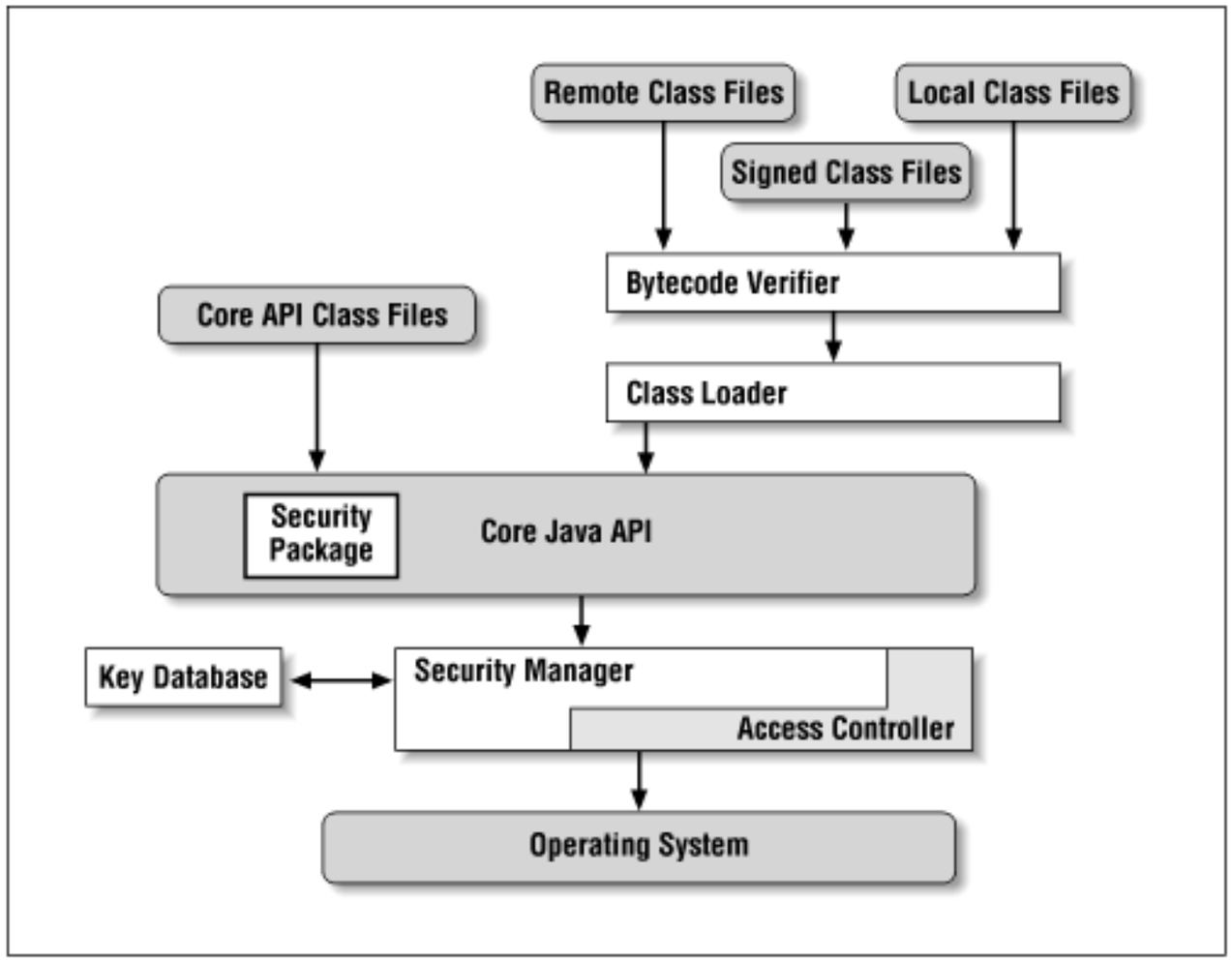
- **Sécurité manager: mécanisme de sandbox (bac à sable)** qui isole l'exécution des applications , vérification des signatures du code

- Classes contenant les algorithmes cryptographiques, authentification et protocoles de communication sécurisés.

Les éléments de base Java

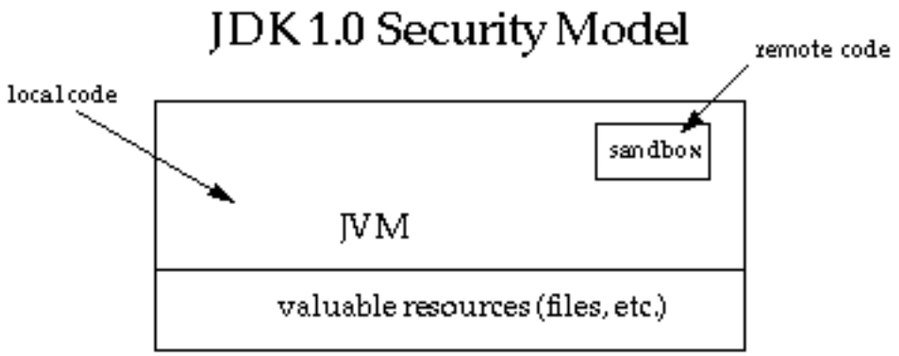
- **Securité de la plateforme**
  - typage fort
  - gestion automatique de la mémoire
  - bytecode verification
  - chargement sécurisé des classes
- **Cryptographie**
- **authentification et contrôle d'accès**
  - Architecture de sécurité
    - notion de sandbox
    - classes pour les permissions
    - access controller
    - class loader
    - security manager
    - security package
    - policytool
    - jarsigner
  - Java Authentification et service d'autorisation
  - Politique de sécurité
  - signatures
- **communications sûres**
- **PKI (Public Key Infrastructure)**

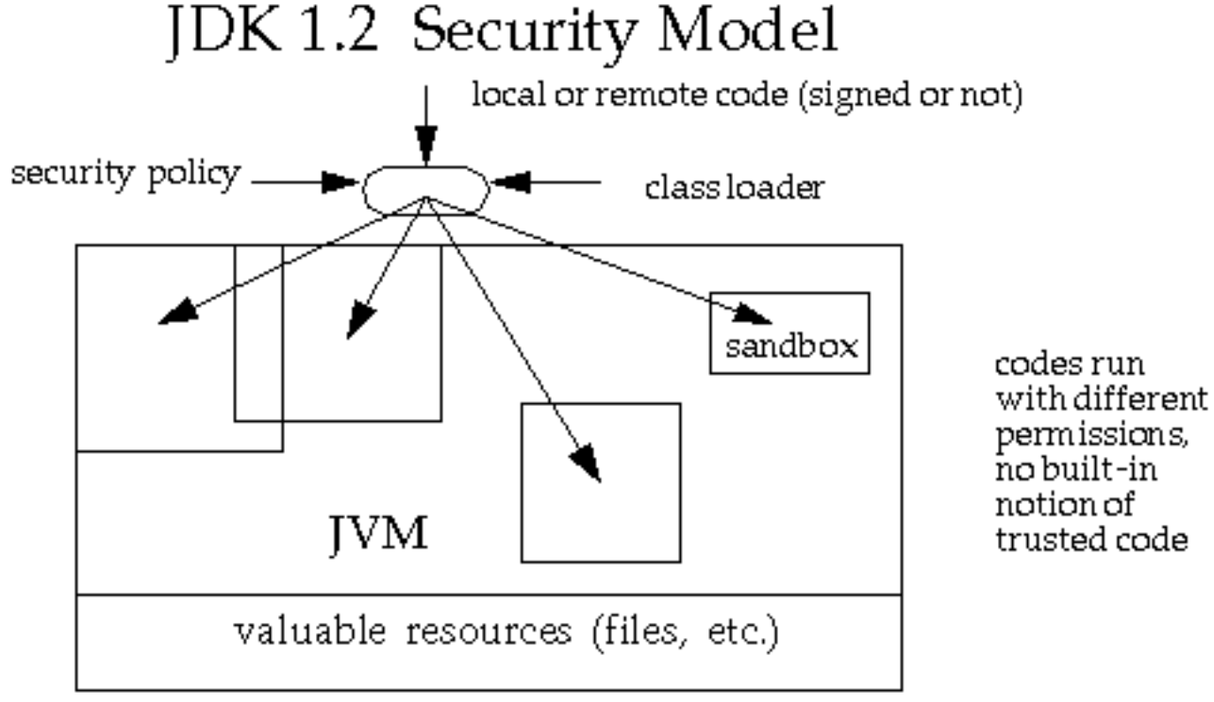
Java


Remote Class Files
Local Class Files
Signed Class Files
Bytecode Verifier
Core API Class Files
Class Loader
Security Package
Core Java API
Key Database
Security Manager
Access Controller
Operating System

Sécurité: sandbox

modèle initial





modèle jdk1.2

Exemple Applet

- une applet s'exécute dans une machine virtuelle java
- une applet *non-signée* est limitée:
  - ne peut pas accéder aux ressources du client (ex: fichiers, clipboard, printer …)
  - ne peut pas se connecter sur des serveurs autres que ceux de son origine
  - ne peut charger des bibliothèques « natives »
  - ne peut pas changer le Security Manager
  - ne peut pas créer de ClassLoader
  - ne peut pas lire certaines propriétés du Système (java.class.path java.home user.dir user.home user.name)
  - (JNLP (java Network Launch Protocol) permet d'augmenter ces droits)

Properties

- Classes Properties
  - extension de Hashtable
    - clé/valeur exemple: `user.home /Users/hf1`
      - `getProperty setProperty`
- `System.getProperty(cle), System.getProperties()`
  - donne ou modifie la valeur d'une « property »

Exemple

```
import java.lang.*;
import java.util.Hashtable;
import java.util.Set;
public class GetProps {
    public static void main(String[] args) {
        String s;
        try {
            System.out.println(" os.name property");
            s = System.getProperty("os.name", "");
            System.out.println("  Nom de l'OS :" + s);
            System.out.println("java.version property");
            s = System.getProperty("java.version", "not specified");
            System.out.println("  version de la JVM : " + s);
            System.out.println("user.home property");
            s = System.getProperty("user.home", "not specified");
            System.out.println(" user home directory: " + s);
            System.out.println("user.dir property");
            s = System.getProperty("user.dir", "not specified");
            System.out.println("  user dir directory: " + s);
            System.out.println("java.path property");
            System.out.println("  java path: " + s);
            s = System.getProperty("java.class.path", "not specified");
            System.out.println("java.home property");
            s = System.getProperty("java.home", "not specified");
            System.out.println("  catalogue de la JRE: " + s);
        } catch (Exception e) {
            System.err.println("exception " + e.toString());
            (System.getProperties()).list(System.out);
        }
    }
}
```

Résultat

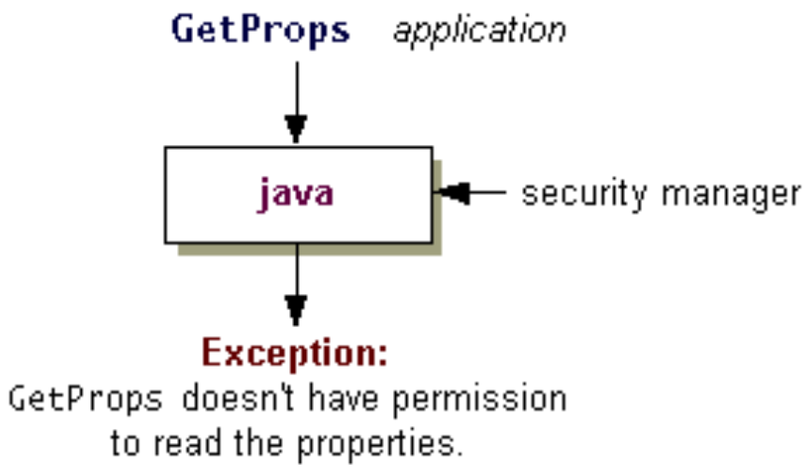
```
 os.name property
   Nom de l'OS :Mac OS X
java.version property
   version de la JVM : 1.8.0
user.home property
  user home directory: /Users/hf1
user.dir property
   user dir directory: /Users/hf1/Netbeans/java/securite
java.path property
   java path: /Users/hf1/Netbeans/java/securite
java.home property
   catalogue de la JRE: /Library/Java/JavaVirtualMachines/
jdk1.8.0.jdk/Contents/Home/jre
```

Sécurité

- sécurité manager (SecurityManager) définit la politique de sécurité pour une application
- par défaut aucun security manager n'est installé: pour l'installer: **`java -Djava.security.manager …`**
- une action non-autorisée lance l'exception SecurityException

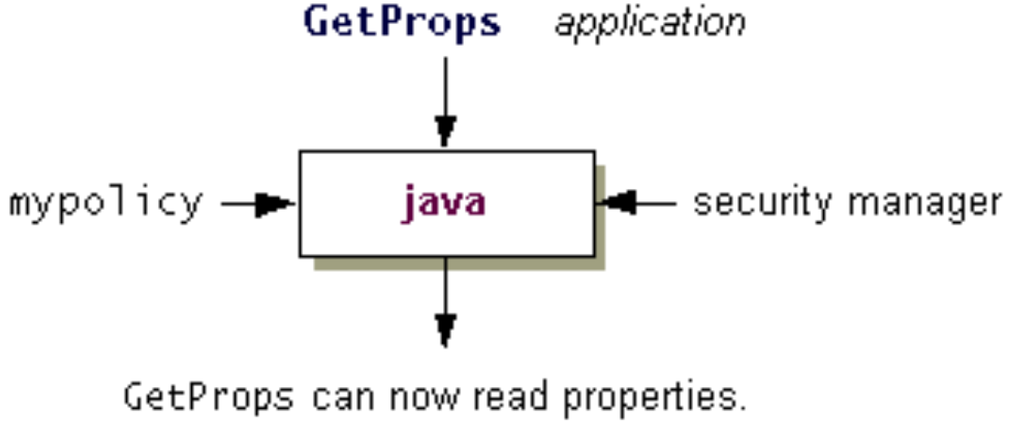
```
SecurityManager sm =System.getSecurityManager();
      if (sm != null) {
          sm.checkXXX(argument,  . . . )
      }
```

lance une exception si XXX n'est pas autorisé
(exemples checkDelete(fichier), checkExit(status))



Security manager

- par défaut, **`user.home`** et **`java.home`** ne sont pas autorisés
- la politique par défaut dans le fichier
  java.home`/lib/security/java.policy`
  java.home`/lib/security/java.security`
  user.home`/.java.policy`
- on peut définir une politique spécifique avec: **`policytool`**



défaut: *java.home*/lib/security/java.policy

```
// Standard extensions get all permissions by default

grant codeBase "file:${java.home}/lib/ext/" {
   permission java.security.AllPermission;
};

// default permissions granted to all domains

grant {
   // allows anyone to listen on un-privileged ports
   permission java.net.SocketPermission "localhost:1024-", "listen";

   // "standard" properties that can be read by anyone

   permission java.util.PropertyPermission "java.version", "read";
   permission java.util.PropertyPermission "java.vendor", "read";
   permission java.util.PropertyPermission "java.vendor.url", "read";
   permission java.util.PropertyPermission "java.class.version", "read";
   permission java.util.PropertyPermission "os.name", "read";
   permission java.util.PropertyPermission "os.version", "read";
   permission java.util.PropertyPermission "os.arch", "read";
   permission java.util.PropertyPermission "file.separator", "read";
   permission java.util.PropertyPermission "path.separator", "read";
   permission java.util.PropertyPermission "line.separator", "read";

   permission java.util.PropertyPermission "java.specification.version", "read";
   permission java.util.PropertyPermission "java.specification.vendor", "read";
   permission java.util.PropertyPermission "java.specification.name", "read";

   permission java.util.PropertyPermission "java.vm.specification.version", "read";
   permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
   permission java.util.PropertyPermission "java.vm.specification.name", "read";
   permission java.util.PropertyPermission "java.vm.version", "read";
   permission java.util.PropertyPermission "java.vm.vendor", "read";
   permission java.util.PropertyPermission "java.vm.name", "read";
};
```

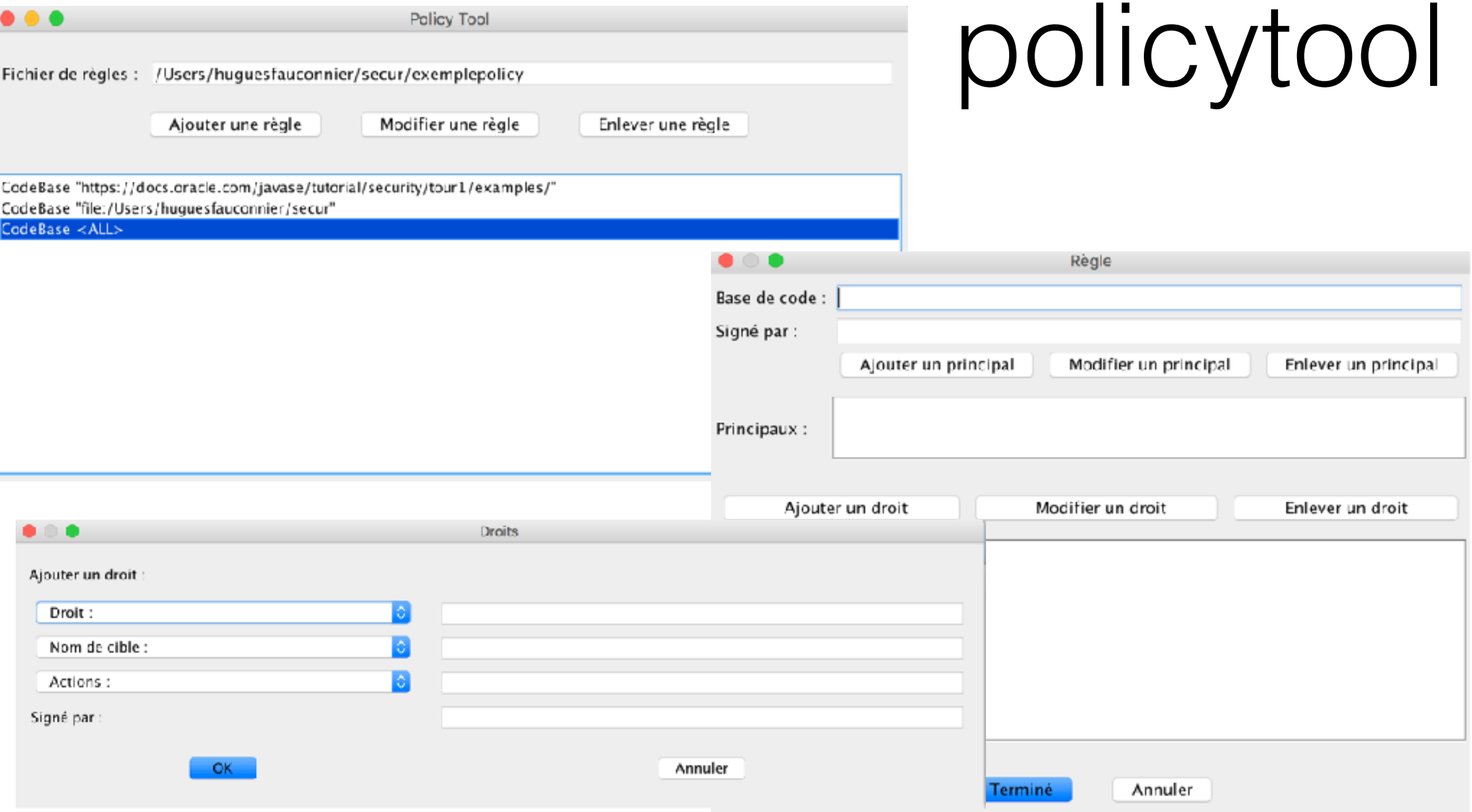
```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=sun.security.ec.SunEC
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
#
# Default login configuration file
#
#login.config.url.1=file:${user.home}/.java.login.config

# whether or not we allow an extra policy to be passed on the command line
# with -Djava.security.policy=somefile. Comment out this line to disable
# this feature.
policy.allowSystemProperty=true

# whether or not we look into the IdentityScope for trusted Identities
# when encountering a 1.1 signed JAR file. If the identity is found
# and is trusted, we grant it AllPermission.
policy.ignoreIdentityScope=false

#
# Default keystore type.
#
keystore.type=jks
# List of comma-separated packages that start with or equal this string
# will cause a security exception to be thrown when
# passed to checkPackageAccess unless the
# corresponding RuntimePermission ("accessClassInPackage."+package) has
# been granted.
package.access=sun.,\
               com.sun.xml.internal.,\
               com.sun.imageio.,\
               com.sun.istack.internal.,\…
     List of comma-separated packages that start with or equal this string
# will cause a security exception to be thrown when
# passed to checkPackageDefinition unless the
# corresponding RuntimePermission ("defineClassInPackage."+package) has
# been granted.
#
# by default, none of the class loaders supplied with the JDK call
# checkPackageDefinition.
#
package.definition=sun.,\
                   com.sun.xml.internal.,\
                   com.sun.imageio.,\
                   com.sun.istack.internal.,\…
```

java.security

résultat de policytool

```
/* AUTOMATICALLY GENERATED ON Wed Feb 18 15:31:26 CET 2015*/
/* DO NOT EDIT */

grant codeBase "http://docs.oracle.com/javase/tutorial/security/tour1/examples/" {
};

grant codeBase "file:/Users/hf1/Netbeans/java/securite/src/securite" {
  permission java.util.PropertyPermission "user.home", "read";
  permission java.util.PropertyPermission "java.home", "read";
};

— — — — — —-
java -Djava.security.manager -Djava.security.policy=maPolitique GetProps
```

policytool




```
java -Djava.security.manager -Djava.security.policy=examplepolicy Appli
```

politique de sécurité

Contenu:

```
grant codeBase "file:${{java.ext.dirs}}/*" {
    permission java.security.AllPermission;
};
```

les fichiers f`ile:${{java.ext.dirs}}/*` auront la permission **java.security.AllPermission** (toutes les autorisations possibles).
Si un sécurité manager est installé, les extensions (jar) dans ces fichiers auront les privilèges nécessaires.
Les codes instances de PrivilegedAction en argument de doPrivileged aussi.

SecurityManager

```
package com.tutorialspoint;

import java.io.FilePermission;
import java.security.AccessControlContext;
import java.security.AccessController;

public class SecurityManagerDemo extends SecurityManager {

   public static void main(String[] args) {
      // le contexte et définir la politique
      AccessControlContext con = AccessController.getContext();
      System.setProperty("java.security.policy", "file:/home/hf/java.policy");
      // créer et utiliser un security manager
      SecurityManagerDemo sm = new SecurityManagerDemo();
      System.setSecurityManager(sm);
      // vérifier l'accès
      sm.checkPermission(new FilePermission("test.txt", "read,write"), con);
      System.out.println("autorisé!");
   }
}
```

java.policy:

```
grant {
  permission java.lang.RuntimePermission "setSecurityManager";
  permission java.lang.RuntimePermission "createSecurityManager";
  permission java.lang.RuntimePermission "usePolicy";
};
```

```
Exception in thread "main" java.security.AccessControlException: access denied (java.io.FilePermission test.txt read,write)
```

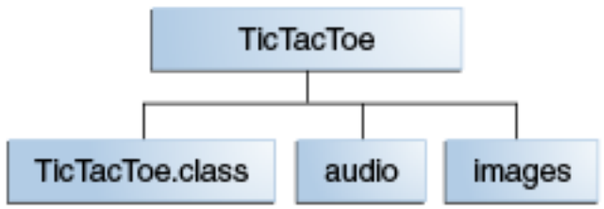
Exemple

```
import java.io.*;
class PasswordSecurityManager extends SecurityManager {
    private String password;
    PasswordSecurityManager(String password) {
        super();
        this.password = password;
    }
    private boolean accessOK() {
        int c;
        DataInputStream dis = new DataInputStream(System.in);
        String response;
        System.out.println("Le password?");
        try {
            response = dis.readLine();
            if (response.equals(password))
                return true;
            else
                return false;
        } catch (IOException e) {
            return false;
        }
    }
    public void checkRead(FileDescriptor filedescriptor) {
        if (!accessOK())
            throw new SecurityException("Raté!");
    }
    public void checkRead(String filename) {
        if (!accessOK())
            throw new SecurityException("No Way!");
    }
    public void checkRead(String filename, Object executionContext) {
        if (!accessOK())
            throw new SecurityException(« C'est foutu!");
    }
    public void checkWrite(FileDescriptor filedescriptor) {
        if (!accessOK())
            throw new SecurityException("Non!");
    }
    public void checkWrite(String filename) {
        if (!accessOK())
            throw new SecurityException( "même pas en rêve!");
    }
}
```

fichiers jar

- créer un fichier jar: **`jar cf jar-file input-file(s)`**
- voir le contenu d'un fichier jar: **`jar tf jar-file`**
- extraire le contenu: **`jar xf jar-file`**
- extraire un contenu: **`jar xf jar-file archived-file(s)`**
- executer: **`java -jar app.jar`**
- applet:

```
<applet code=AppletClassName.class
        archive="JarFileName.jar"
        width=width height=height>
</applet>
```



Exemples

- exemples:

```
jar cvf TicTacToe.jar TicTacToe.class\
    audio images
(Créer)

jar tvf TicTacToe.jar
(contenu)

jar xf TicTacToe.jar TicTacToe.class \
    images/cross.gif
(eXtraire)

jar uf TicTacToe.jar images/new.gif
(update)

(java -jar app.jar exécution)
```

manifest

- définit les fonctionnalités de l'archive
- création par défaut:
  META-INF/MANIFEST.MF

  Manifest-Version: 1.0
  Created-By: 1.7.0_06 (Oracle Corporation)

- **`jar cfm jar-file manifest-addition input-file(s)`**
  pour changer le contenu du manifest

MANIFEST

```
Exemple:
   Manifest-Version: 1.0
   Ant-Version: Apache Ant 1.9.4
   Created-By: 1.8.0-b132 (Oracle Corporation)
   Class-Path:
   X-COMMENT: Main-Class will be added automatically by build
   Main-Class: securite.GetProps
```

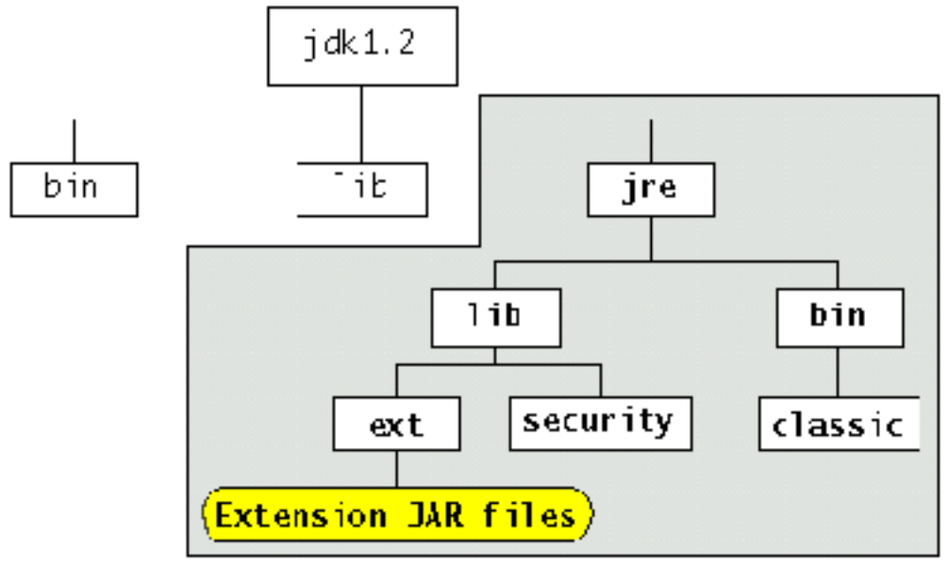
Main-Class est le point d'entrée pour exécuter l'application
Class-Path: pour contenir des extensions .jar

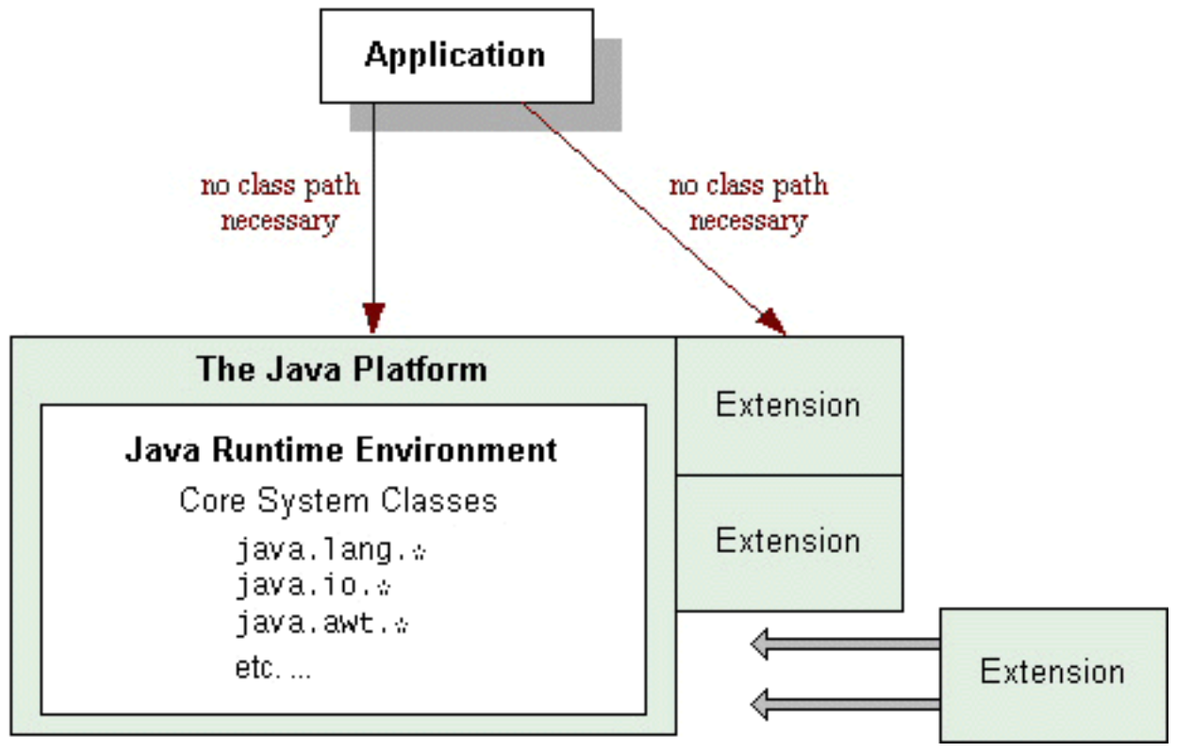
On peut changer le point d'entrée:

```
jar cfe app.jar MyApp MyApp.class
jar cfe Main.jar foo.Main foo/Main.class
```

Extensions

- structure du jdk
- ext contient des (.jar) extensions que l'on peut ajouter à l'environnement de le JVM
- (alternative:
  `java -classpath ext.jar appli`
)





Extensions et sécurité

```
import java.io.*;
import java.security.*;

public final class MaClass {
 public static void
   maFonc(final   Param r) {
   AccessController.doPrivileged(
              new PrivilegedAction() {
                 public Object run() {…
                // nécessite des privilèges
              }
       });
    }
}
```

code dans la méthode run d'un objet java.security.PrivilegedAction

méthode doPrivileged appliquée à la PrivilegedAction rend ce code privilégié

(sinon les permissions sont les permissions minimales de la chaîne des appels)

AccessController

- AccessController
- méthodes:
  - checkPermission

```
FilePermission perm = new FilePermission("/temp/testFile", "read");
AccessController.checkPermission(perm);
```

  - doPrivileged

```
somemethod() {
     ...normal code here...
     AccessController.doPrivileged(new PrivilegedAction<Void>() {
         public Void run() {
             // privileged code goes here, for example:
             System.loadLibrary("awt");
             return null; // nothing to return
         }
     });
     ...normal code here...
 }
```

```
public void doStuff() {

    try {
        /* exception si pas de permission pour l'appelant        */
        System.out.println(System.getProperty("java.home"));
    } catch (Exception e1) {
        System.out.println(e1.getMessage());
    }
    AccessController.doPrivileged(new PrivilegedAction<Boolean>() {
        public Boolean run() {
            try {
                /*
                 * ok si la classe a la permission même
                *  si l'appelant ne l'a pas
                 */
                System.out.println(System.getProperty("java.home"));
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }

            return Boolean.TRUE;
        }
    })
```

doPrivileged

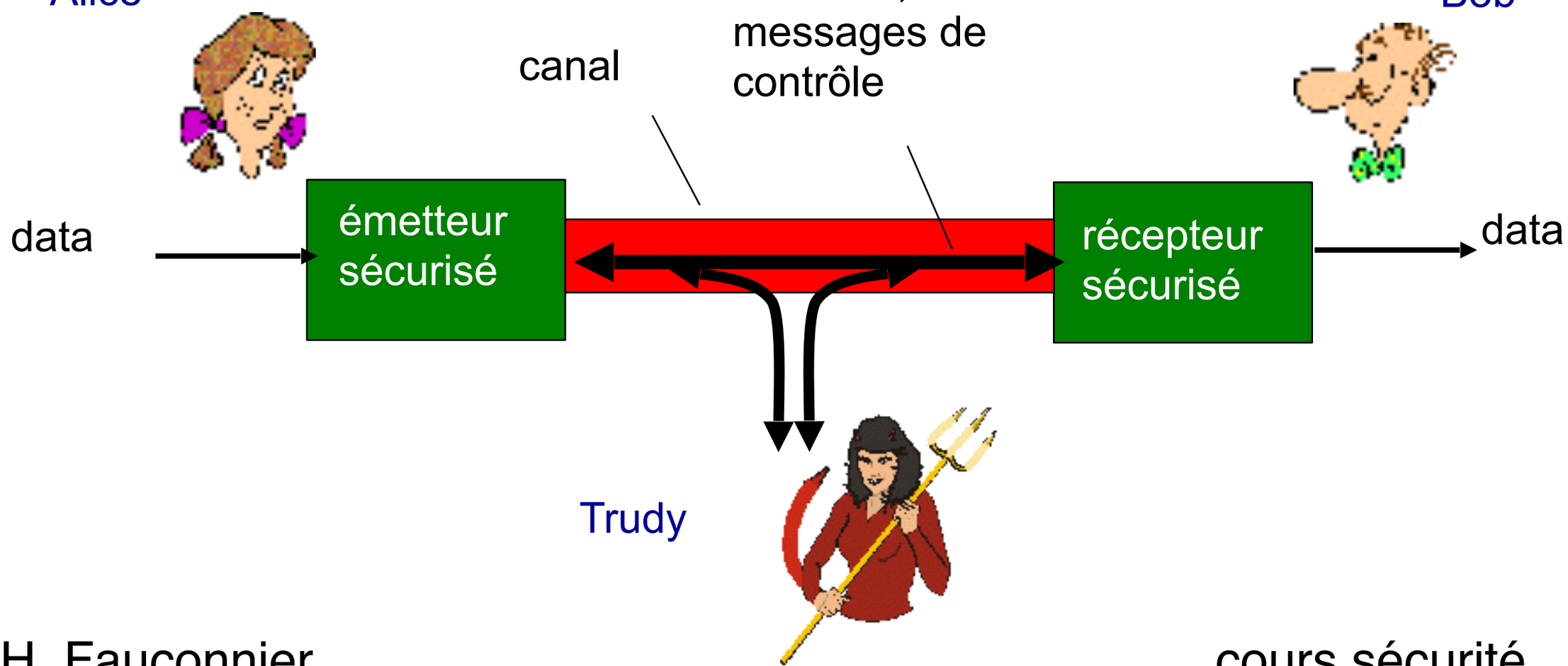
```
grant codeBase "file:/home/somebody/classb.jar" {
    permission java.util.PropertyPermission "java.home", "read";
};
```

doStuff() est défini dans classb et appelé par classa

Quelques éléments de cryptographie

Modèle de base pour la communication

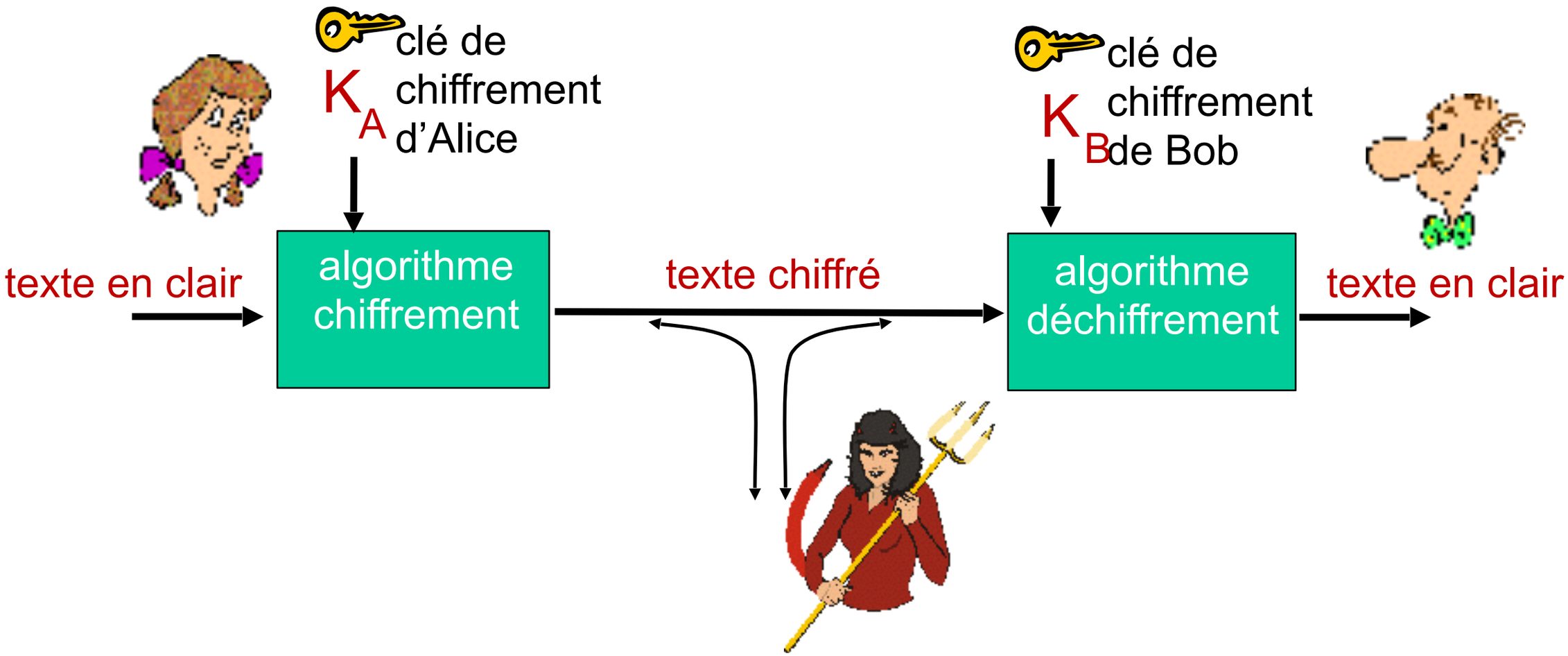
- Bob, Alice veulent communiquer se façon sûre
- Trudy (l'intruse) peut intercepter, supprimer ajouter des messages



Que peut faire un bad guy (réseau)

- *espionner (eavesdrop):* intercepter des messages
- *insérer* des messages
- *imposture (impersonation):* mettre de fausses adresses sources dans les paquets (parodie - spoof)
- *pirater (hijacking) :* prendre la place de l'émetteur ou du récepteur
- *déni de service*: empêcher le service de fonctionner (surcharge des ressources)

Cryptographie: chiffrage / déchiffrage

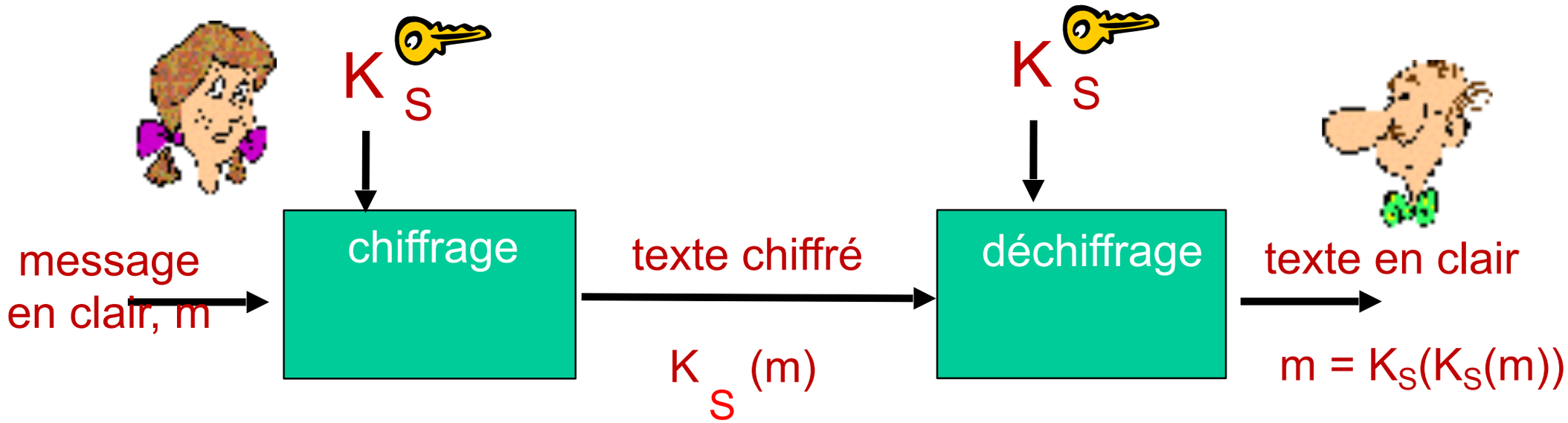


m message en clair

$K_A(m)$ message chiffré avec la clé K_A

$m = K_B(K_A(m))$

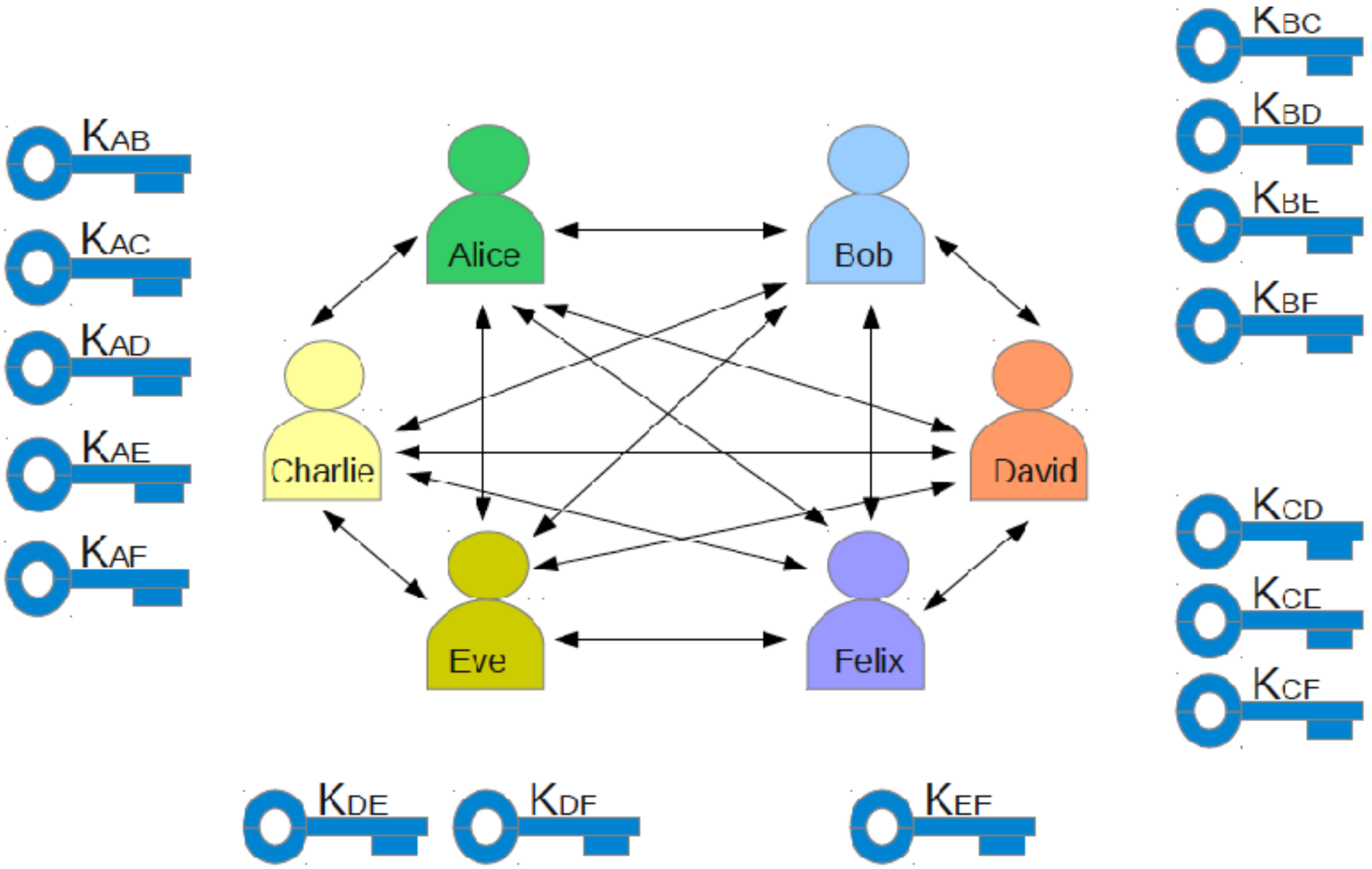
Cryptographie symétrique



cryptographie à clés symétriques: Bob et Alice partagent la même clé (symétrique) K_S

Problème: Comment Bob et Alice obtiennent la clé?

Des clefs:


KAB
KAC
KAD
KAE
KAF
Alice
Bob
Charlie
David
Eve
Felix
KBC
KBD
KBE
KBF
KCD
KCE
KCF
KDE
KDF
KEF

Plus élaboré

- n codes à substitution: $M_1, M_2, \ldots, M_n$
- motifs cycliques:
  - exemple n=4: M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; ..
- pour chaque nouveau symbole utiliser cycliquement le motif de substitution suivant:
  - dog: d de M_1, o de M_3, g de M_4

Clé de chiffrement: n codes à substitution, et un motif de substitution

- la clé n'est pas seulement un motif de n bits

DES: chiffrement symétrique

DES: Data Encryption Standard

- US standard [NIST 1993]
- Clef symétrique 56-bit, texte en clair de 64-bit
- chiffrage par blocs avec chainage des chiffrages
- DES est-il sûr?
  - DES Challenge: 56-bit-key-encrypted phrase déchiffrée(brute force brute) en moins d'un jour
  - pas de « bonne » attaque analytique connue
- DES plus sûr:
  - 3DES: chiffrement 3 fois avec 3 clés de chiffrement

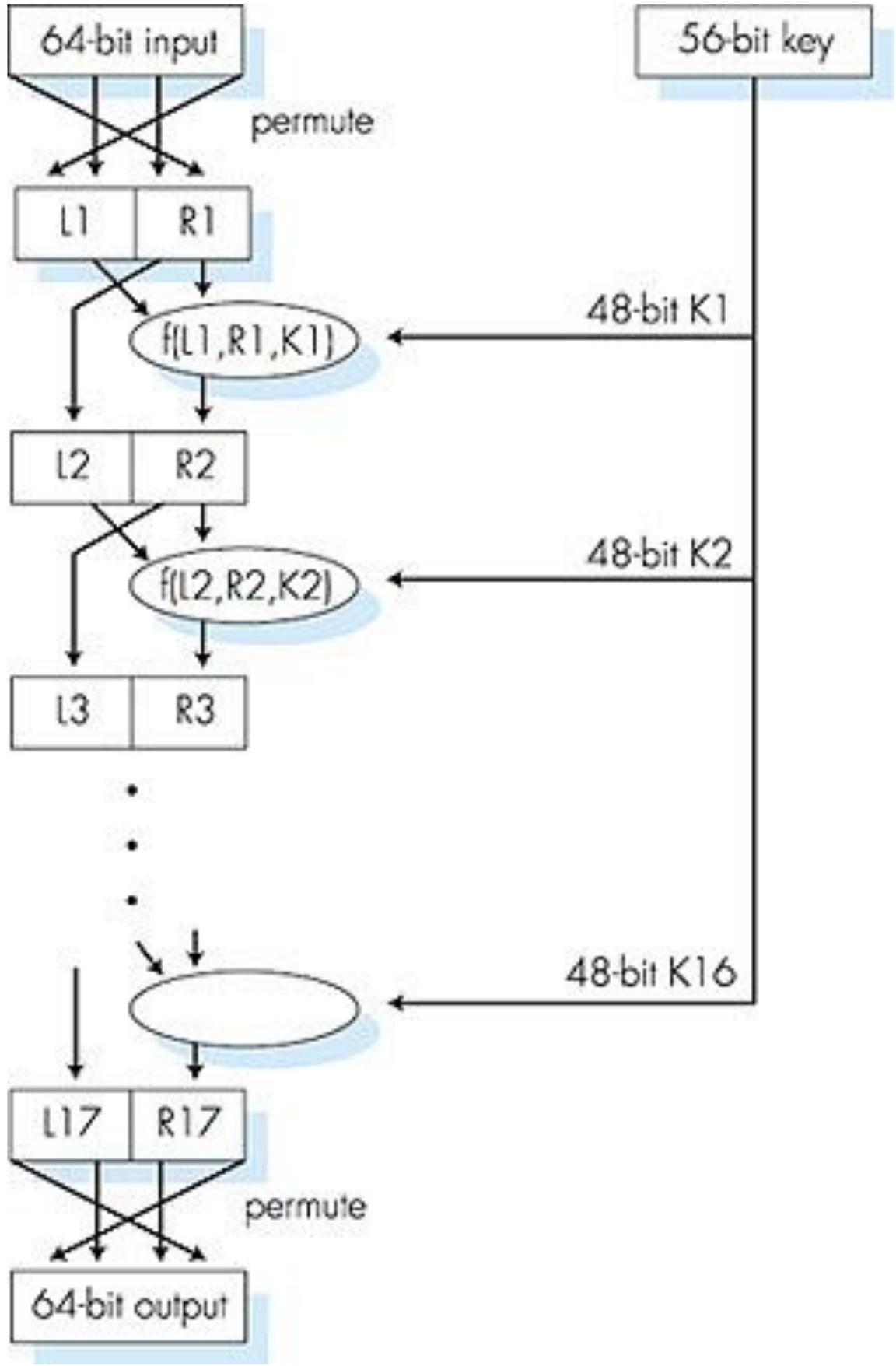
DES: code symétrique

DES:

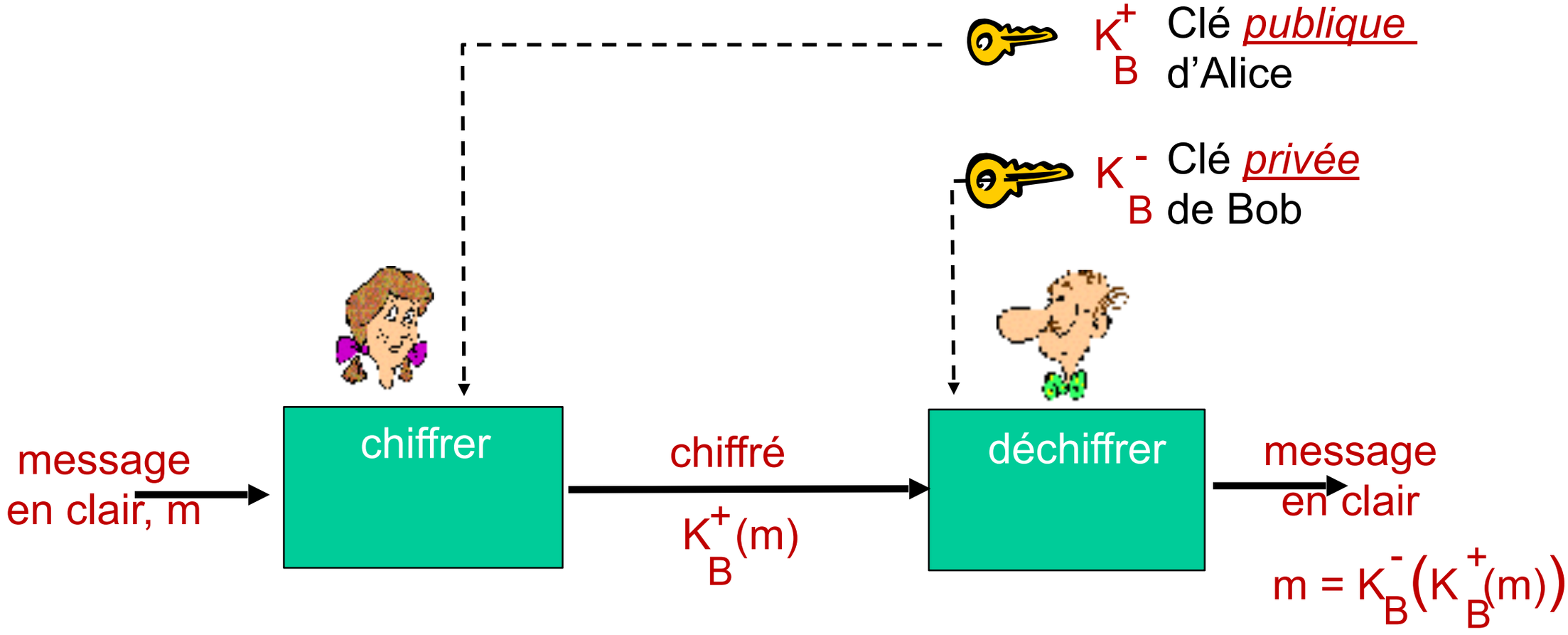
permutation initiale

16“rondes” identiques d’application de fonctions, chacune utilisant 48 bits différents de la clé

permutation finale



Cryptographie à clés publiques (Cryptographie asymétrique)



Algorithme de chiffrement

Principe:

1. $K_B^+(\,)$ et $K_B^-(\,)$ vérifient

$$K_B^-(K_B^+(m)) = m$$

2. A partir de la clé publique K_B^+, il est « impossible » de calculer la clé privée K_B^-

($K_B^+()$ et $K_B^-()$ sont inverses l'une de l'autre, $K_B^+()$ est facile à calculer mais à partir de $K_B^+()$ il est très difficile de calculer $K_B^-()$ *one-way functions)*

RSA: Rivest, Shamir, Adelson

RSA: création des clés privée/publique

1. choisir deux grands nombres premiers *p* et *q* (par exemple 1024 bits)
2. calculer $n = pq$, $z = (p-1)(q-1)$
3. choisir *e* $(e<n)$ sans diviseur commun avec z (*e, z* sont premiers entre eux).
4. choisir *d* tel que *ed-1* est divisible par *z*. (*ed* mod $z = 1$).
5. *clé publique* $\underbrace{(n,e)}_{K_B^+}$. *clé privée* $\underbrace{(n,d)}_{K_B^-}$.

RSA: chiffrement, déchiffrement,

0. soit (n,e) et (n,d) obtenus précédemment

1. message m ($<n$), c le message chiffré:
   $c = m^e \bmod n$

2. pour déchiffrer c:
   $m = c^d \bmod n$

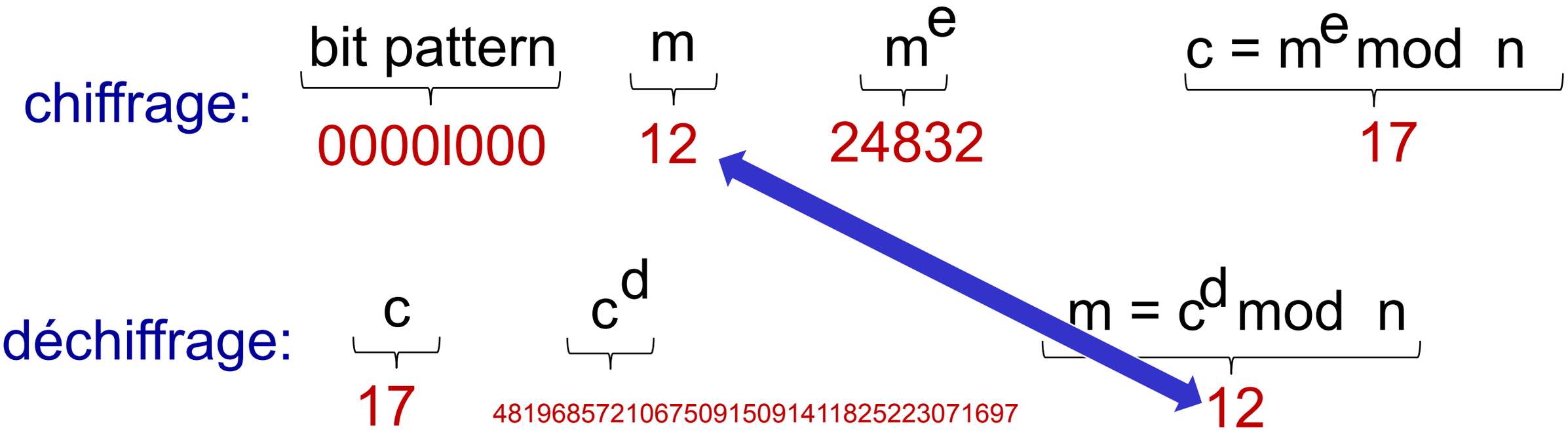
On a $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

RSA exemple:

Bob choisit *p=5, q=7*. Alors *n=35, z=24*.

e=5 (*e, z* premiers entre eux).
d=29 (*ed-1* divisible par z).

chiffrement message de 8-bits



Chiffrement à clés publiques

RSA vérifie aussi:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{d'abord la clé publique ensuite la clé privée}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{d'abord la clé privée ensuite la clé publique}}$$

utile pour le signatures


Anne
CléA$_{privé}$
Seule Anne peut avoir écrit ceci
CléA$_{publique}$
Bob
CléA$_{publique}$
Dylan


Anne
CléA$_{privé}$
Seule Anne peut lire ceci
CléA$_{publique}$
Bob
CléA$_{publique}$
Dylan

Pourquoi RSA est sûr?

- A partir de la clé publique de Bob (n,e). Il est difficile de trouver d
- « nécessite » de factoriser n sans connaître p et q.
  - On considère que factoriser un grand nombre est difficile.

RSA dans la réalité

- l'exponentiation utilisée dans RSA est coûteuse
- DES est 100 fois plus rapide que RSA
- En pratique:
  - utiliser un système à clés publiques pour établir une communication sûre et s'entendre sur une clé symétrique, utiliser cette clé pour chiffrer-déchiffrer les communications.

Clé de session, K_S

- Bob et Alice utilisent RSA pour échanger une clé symétrique K_S
- avec K_S, codage symétrique

Signatures numériques (avec clés asymétriques)

technique analogue à la signature manuelle:

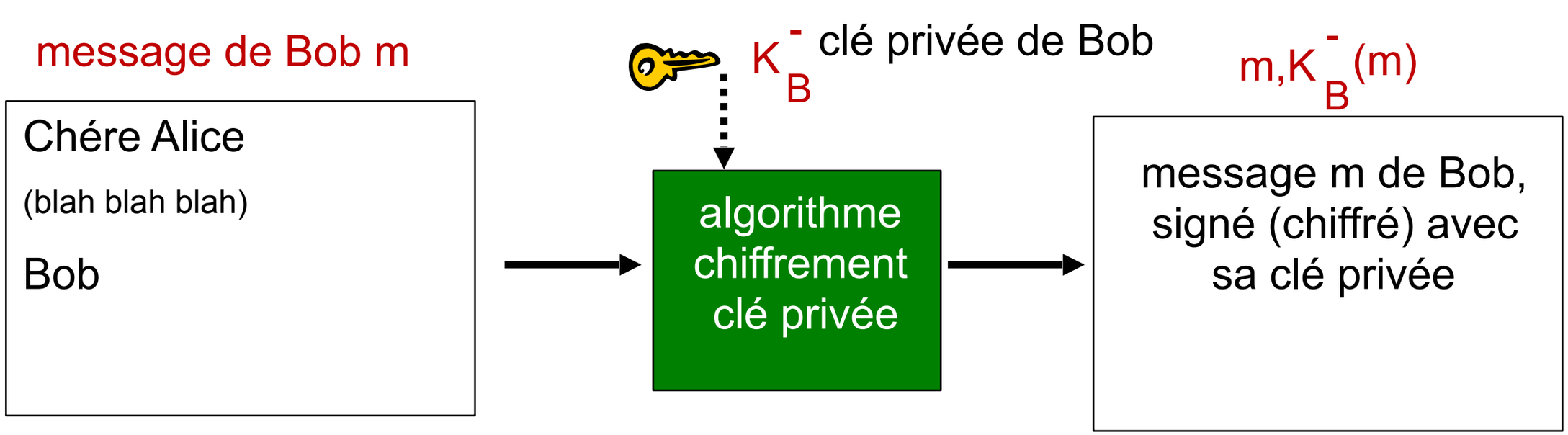
- Bob signe numériquement le document, établissant ainsi qu'il est le créateur/propriétaire du document. ((Bob) peut chiffrer le document avec sa clé privée)
- *vérifiable, infalsifiable (unforgeable):* Alice peut prouver à un tiers que personne d'autres que Bob n'a signé le document. (En chiffrant avec la clé publique de Bob, on doit avoir le document).

- attention… on suppose ici que « tout le monde » sait que la clé publique de Bob est bien la clé publique de Bob et seul Bob connaît sa clé privée

Signatures numériques

simple signature numérique pour le message m:

- Bob signe m en le codant avec sa clé privée K_B^-, créant le message signé , $K_B^-(m)$



Signatures numériques

- si Alice reçoit m, avec la signature: m, $K_B^-(m)$
- Alice vérifie que m est signé par Bob avec la clé publique de Bob K_B^+ : $K_B^+(K_B^-(m)) = m$.
- Si $K_B^+(K_B^-(m)) = m$, celui qui a signé avait la clé privée de Bob

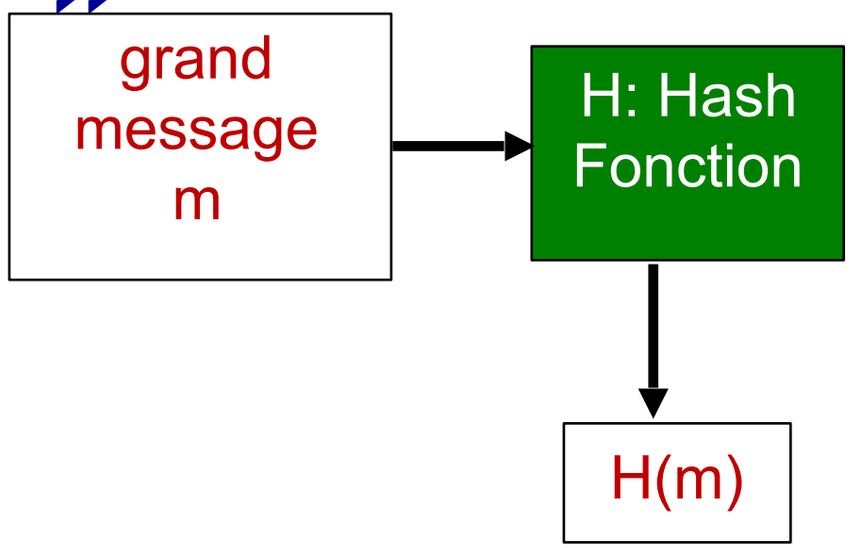
Alice vérifie:

- Bob a signé m
- personne d'autre n'a signé m
- Bob a signé m et pas m'

non-répudiation:

- ✓ Alice peut aller en justice prendre m, et la signature $K_B^-(m)$ et prouver que Bob a signé m

« Message digest »



le chiffrement de longs messages avec clés publique est très coûteux

Mais: mais on peut facilement chiffrer des empreintes (digest) de taille fixe (“fingerprint”)

- en appliquent H fonction de hachage à m, on obtient un digest *H(m).*

Propriétés des fonctions de hachage:

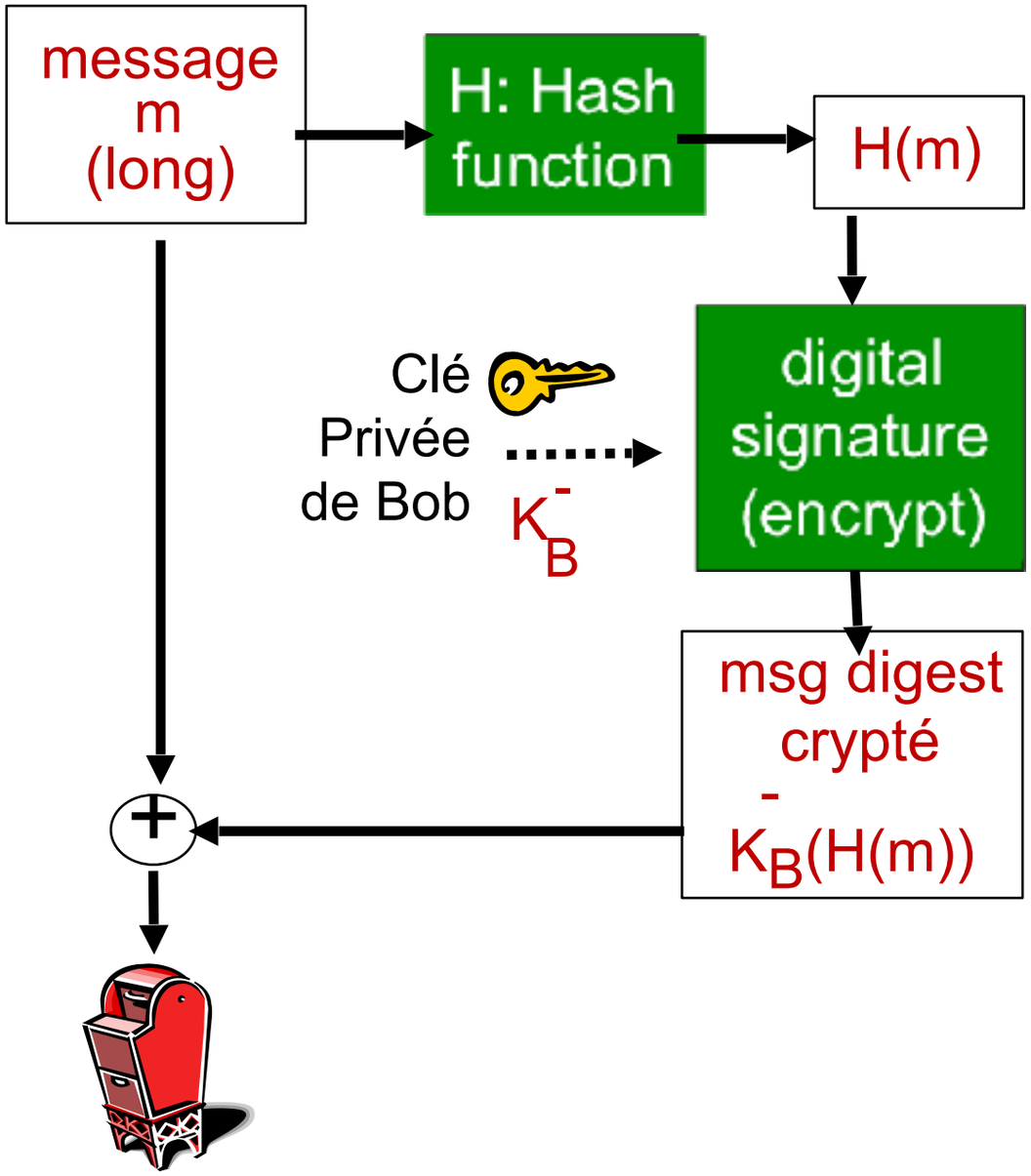
- E=H(M): « impossible » de trouver M connaissant E
- connaissant M et E « impossible »de trouver M’ tel que H(M’)=H(M)=E
- « impossible » de trouver M et M’ tels que H(M)=H(M’)

algorithmes de Hachage

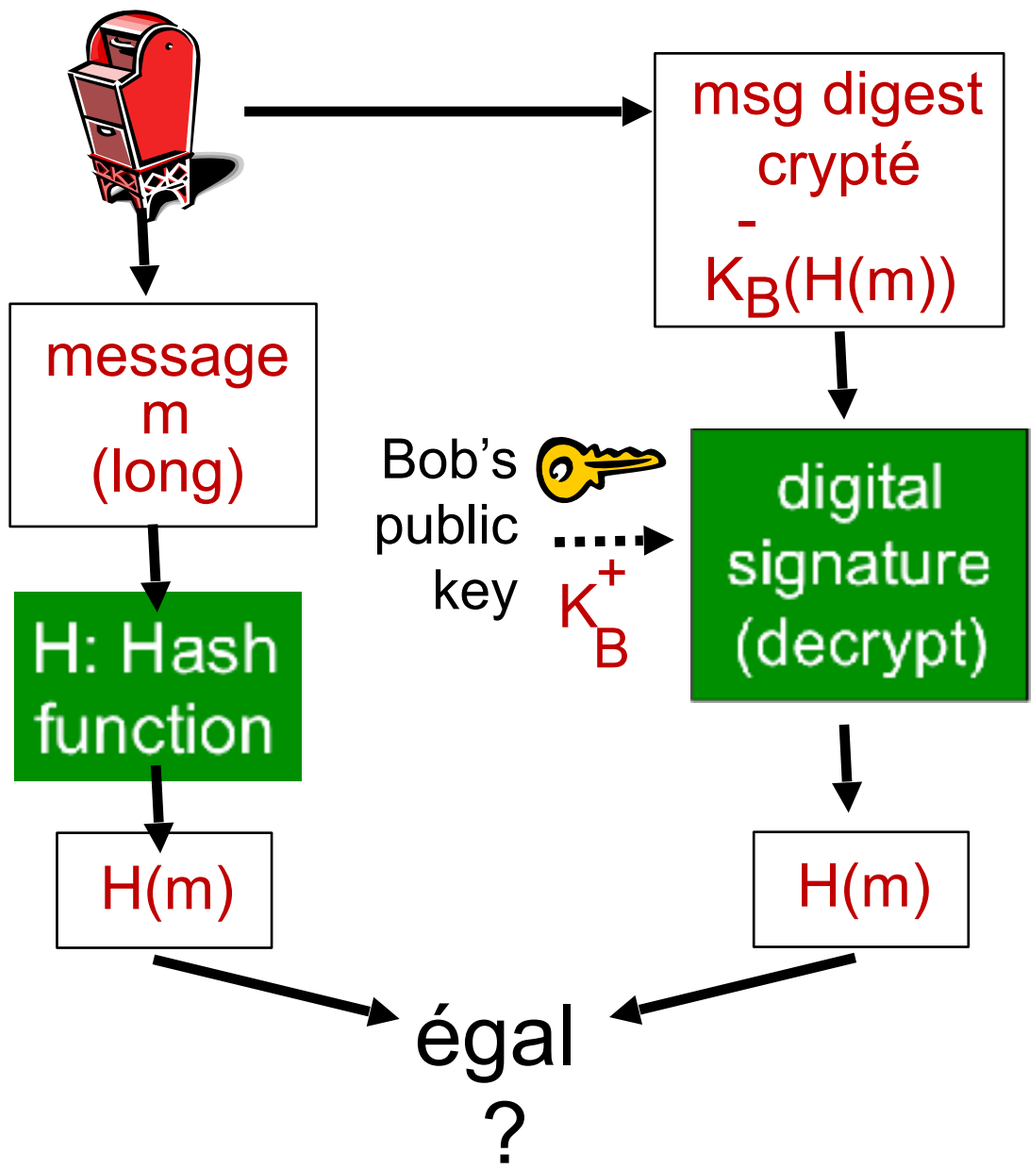
- MD5 (RFC 1321)
  - calcule un « digest » de128-bit
  - à partir d'une chaîne x de128-bits, il est difficile de construire un msg m pour lequel le hachage par MD5 est égal à x
  - « cassé » en 2004
- SHA-256 SHA-512

Digest signé comme signature numérique

Bob envoie le message signé :

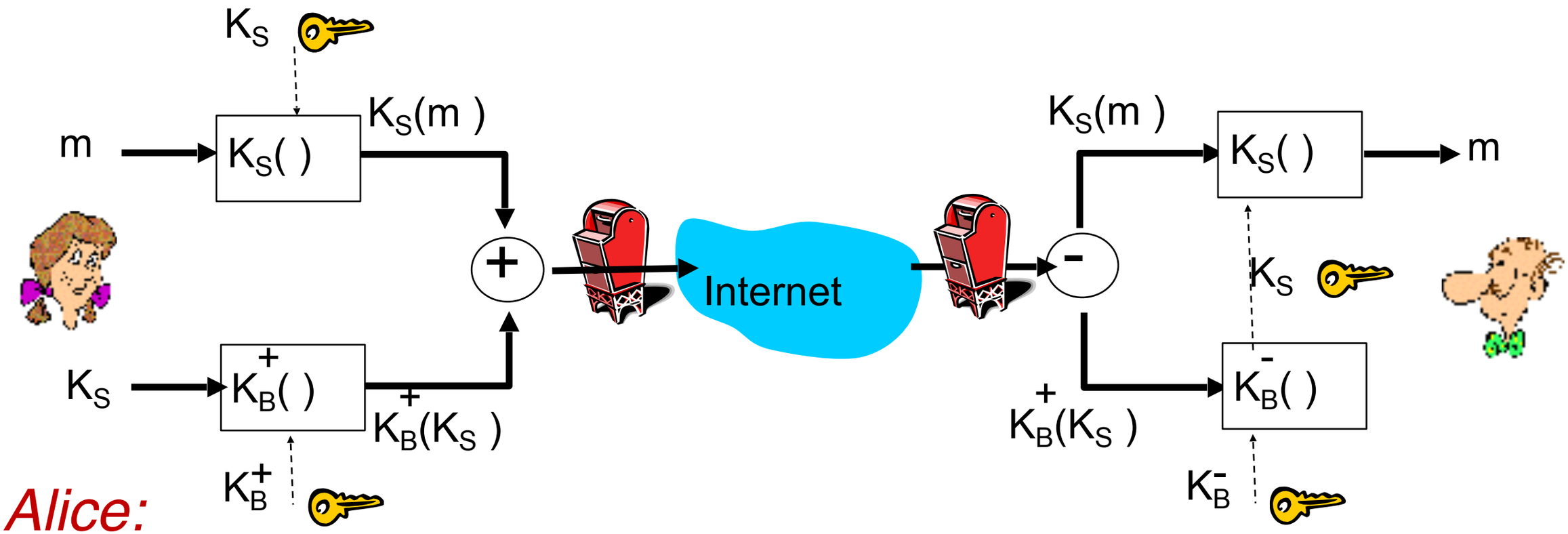


Alice vérifie la signature, l'intégrité du message signé:



e-mail sécurisé

- Alice veut envoyer un mail confidentiel à Bob

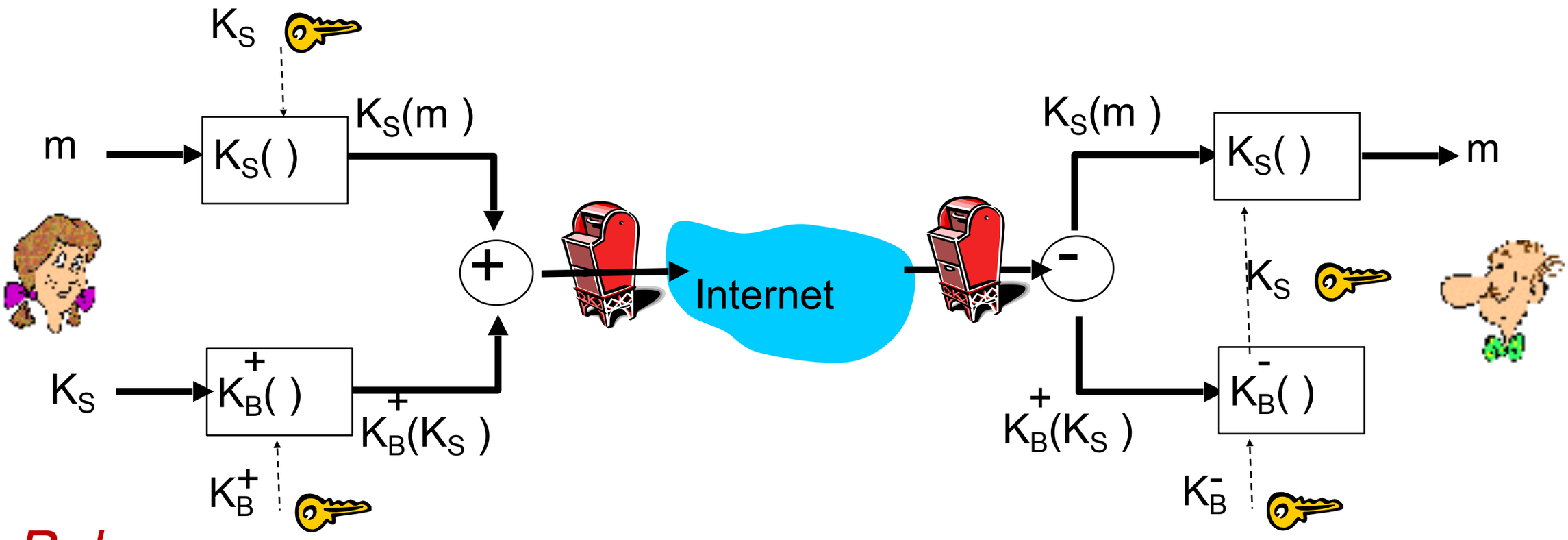


Alice:

- génère une clé *symétrique* privée, K_S
- chiffre le message avec K_S (pour l'efficacité)
- chiffre aussi K_S avec la clé publique de Bob
- envoie $K_S(m)$ et $\overset{+}{K}_B(K_S)$ to Bob

e-mail sécurisé

- Alice veut envoyer un mail confidentiel m à Bob.

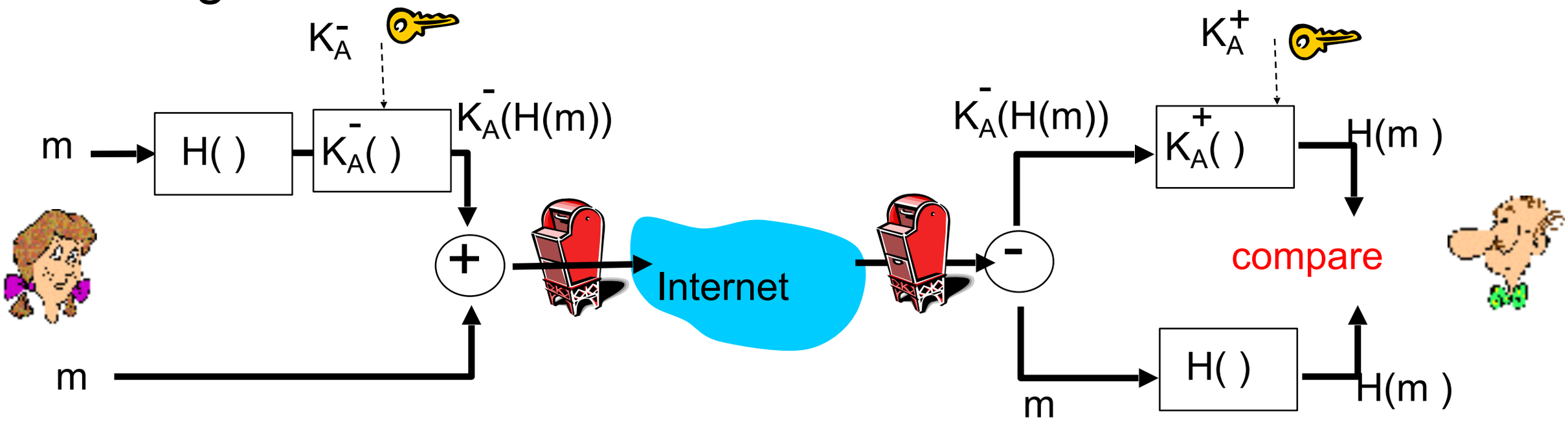


Bob:

- utilise sa clé privée pour déchiffrer et obtient K_S
- utilise K_S pour déchiffrer $K_S(m)$ pour obtenir m

e-mail sécurisé (suite)

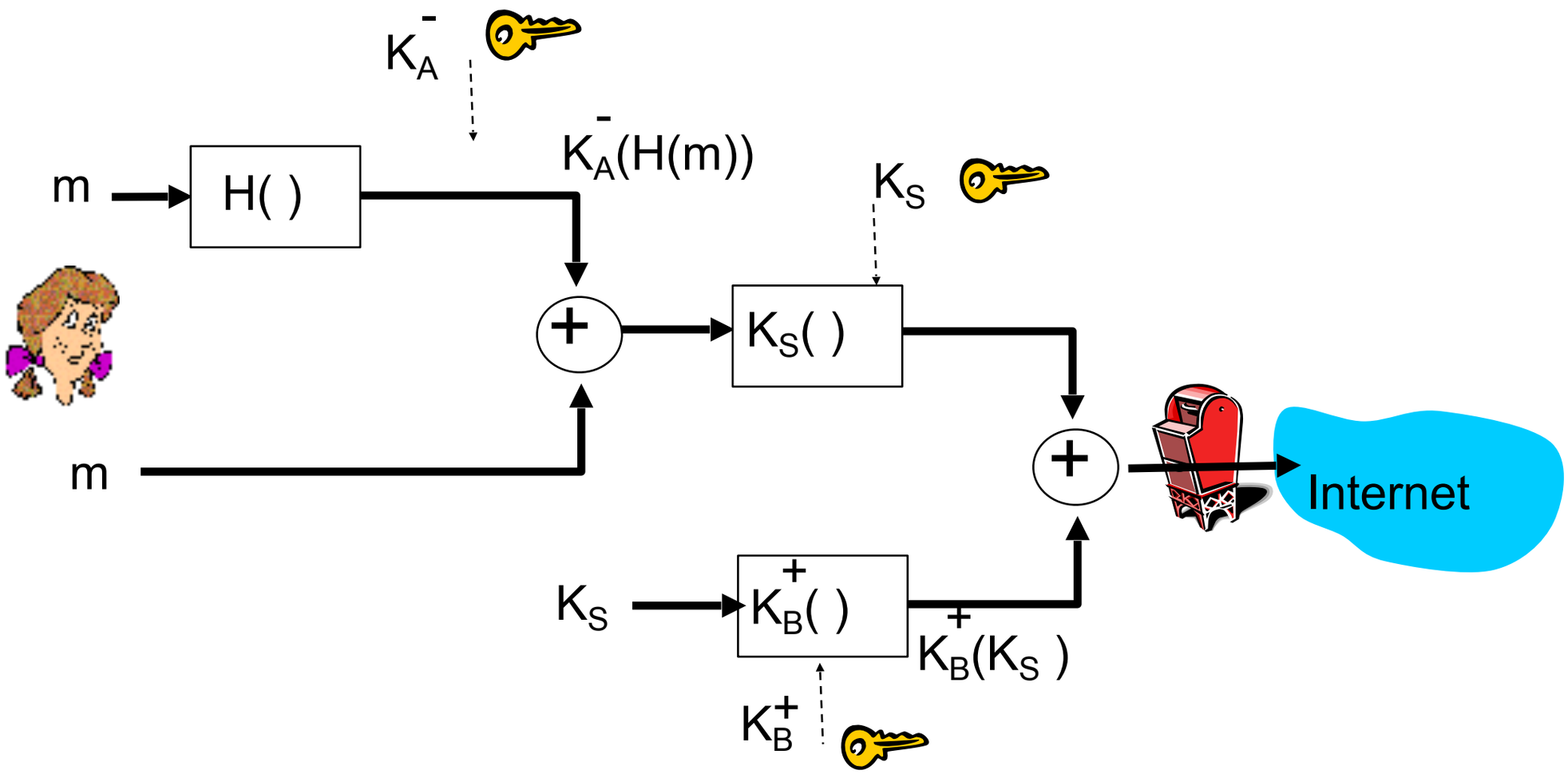
- Alice veut en plus prouver l'intégrité et authentifier le message



- Alice signe le message (signature numérique)
- envoie à la fois le message en clair et la signature numérique

e-mail sécurisé (suite)

- Alice veut tout: secret authentification de l'émetteur et intégrité



Alice utilise 3 clés: sa clé privée, la clé publique de Bob, et une nouvelle clé symétrique

Authentification

- Le système de signature numérique présenté ci-dessus montre que le message a bien été signé par quelqu'un qui possédait une clé privée correspondant à la clé publique attribuée à Bob (attribuée?)
- On peut donc supposer que le message a été signé par Bob… si la clé publique attribuée à Bob est bien celle de Bob
- Mais comment savoir si c'est réellement la clé publique de Bob?
- Comment assurer que celui qui émet le message

Authentification

Goal: Bob veut que Alice lui prouve son identité (Bob veut bien exécuter un code s'il provient d'Alice)

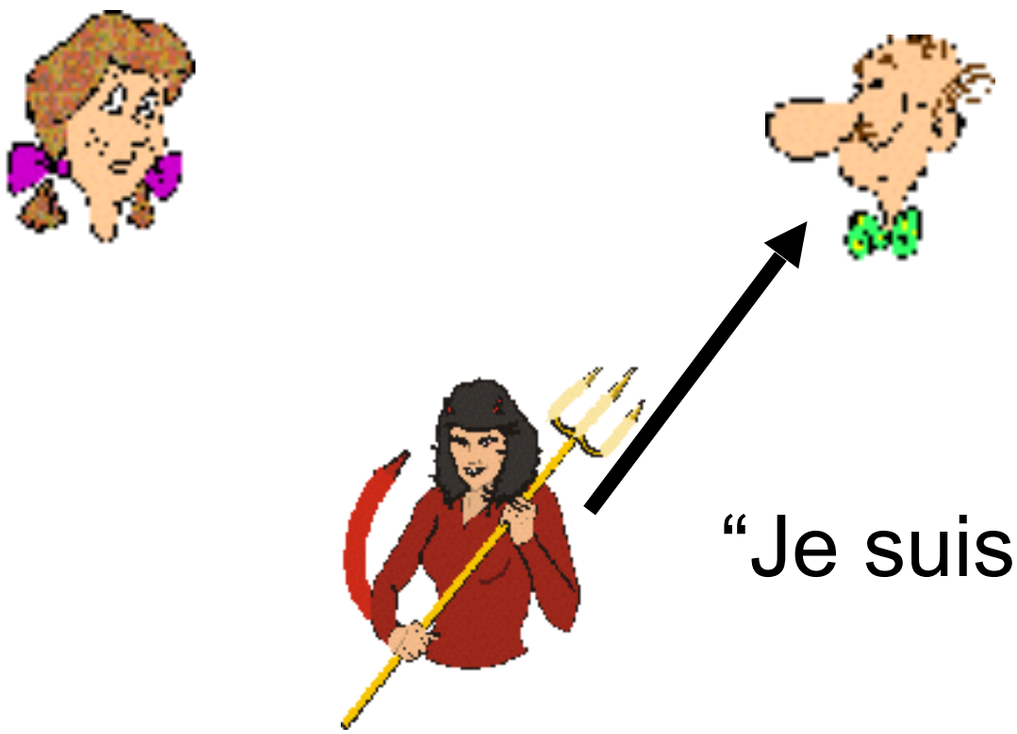
Protocole ap1.0: Alice dit "Je suis Alice"

Quelle faille ?

Authentification

Goal: Bob veut que Alice lui prouve son identité (Bob veut bien exécuter un code s'il provient d'Alice)

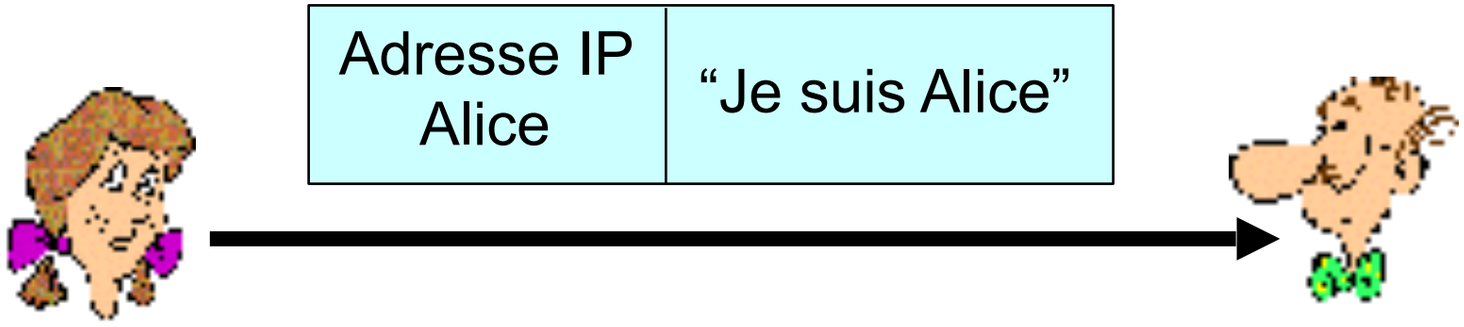
Protocole ap1.0: Alice dit "Je suis Alice"



Bob ne voit pas Alice,
Trudy peut prétendre être Alice

Authentification: autre essai

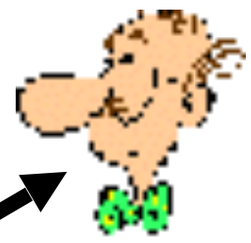
Protocole ap2.0: Alice dit “je suis Alice”
dans un paquet IP avec son adresse IP



La faille??

Authentification: autre essai

Protocole ap2.0: Alice dit “je suis Alice”
dans un paquet IP avec son adresse IP

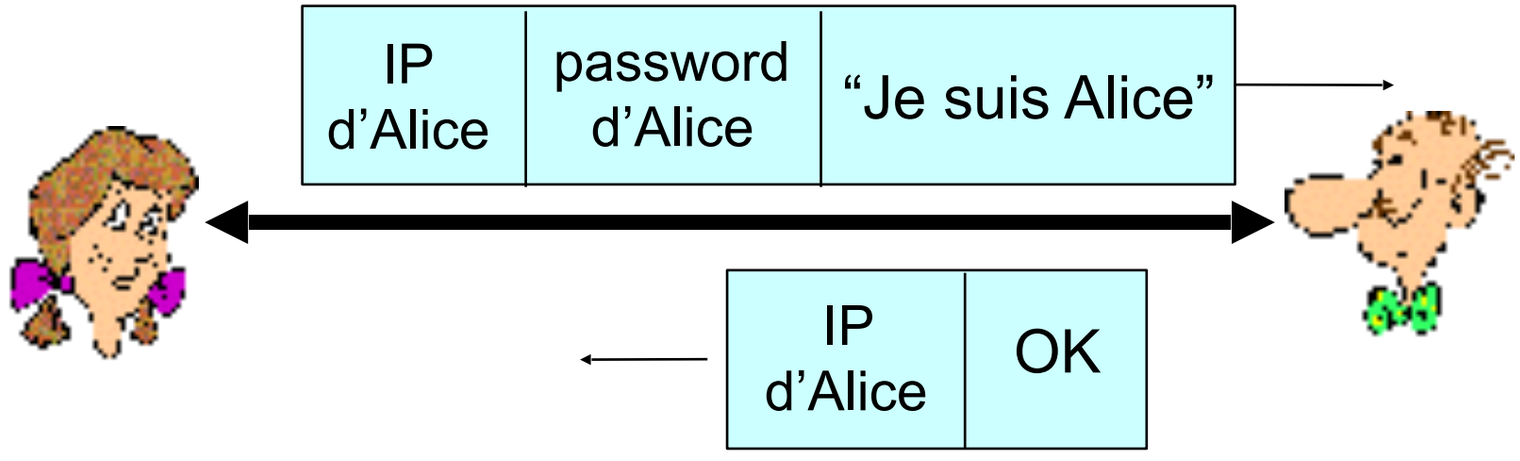


adresse IP d’Alice	“Je suis Alice”

Trudy peut créer un paquet IP en “spoofant” l’adresse d’Alice (c’est possible)

Authentification: autre essai

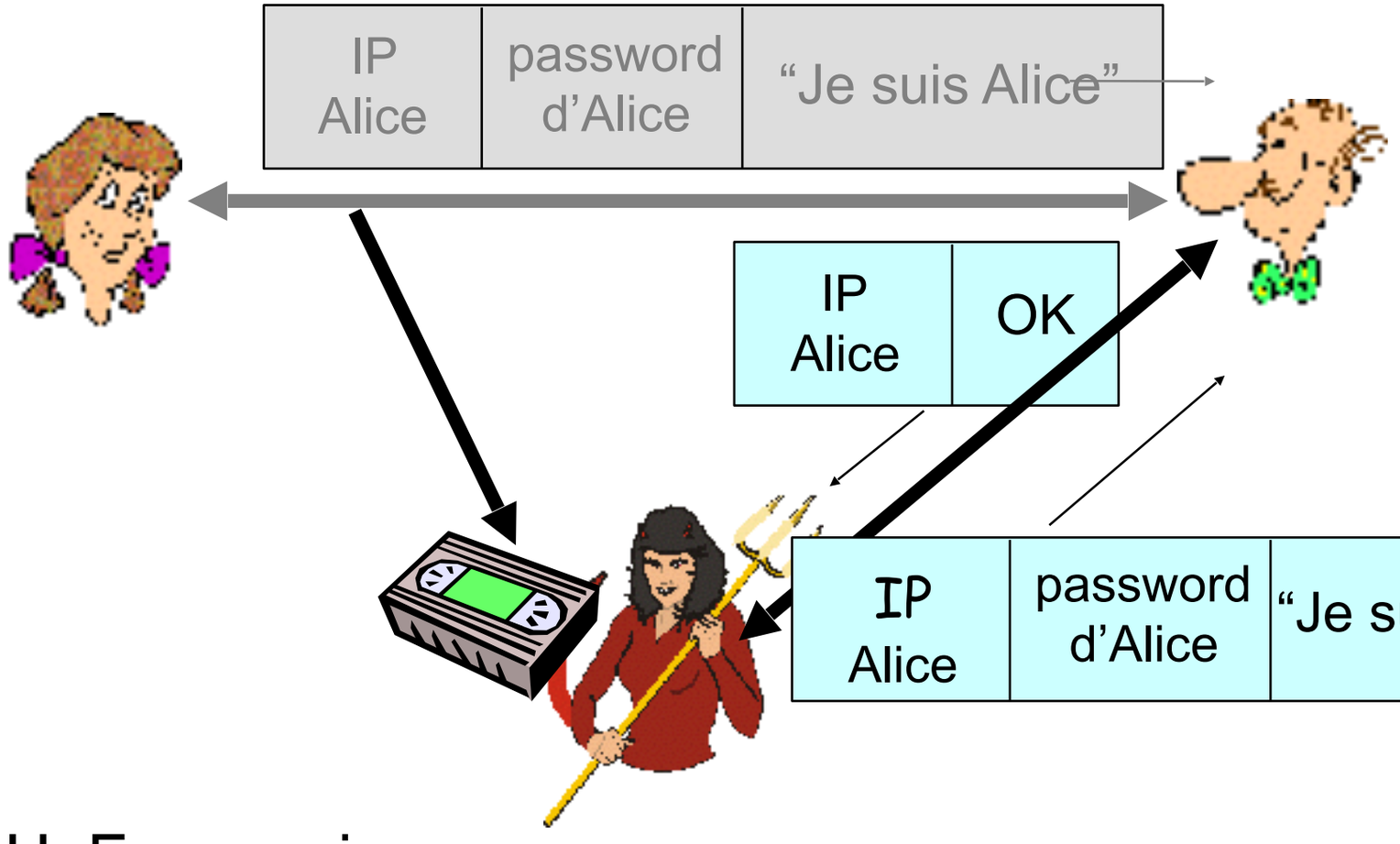
Protocole ap3.0: Alice dit “Je suis Alice” et envoie son password pour le prouver.



Faiblesse ??

Authentification: autre essai

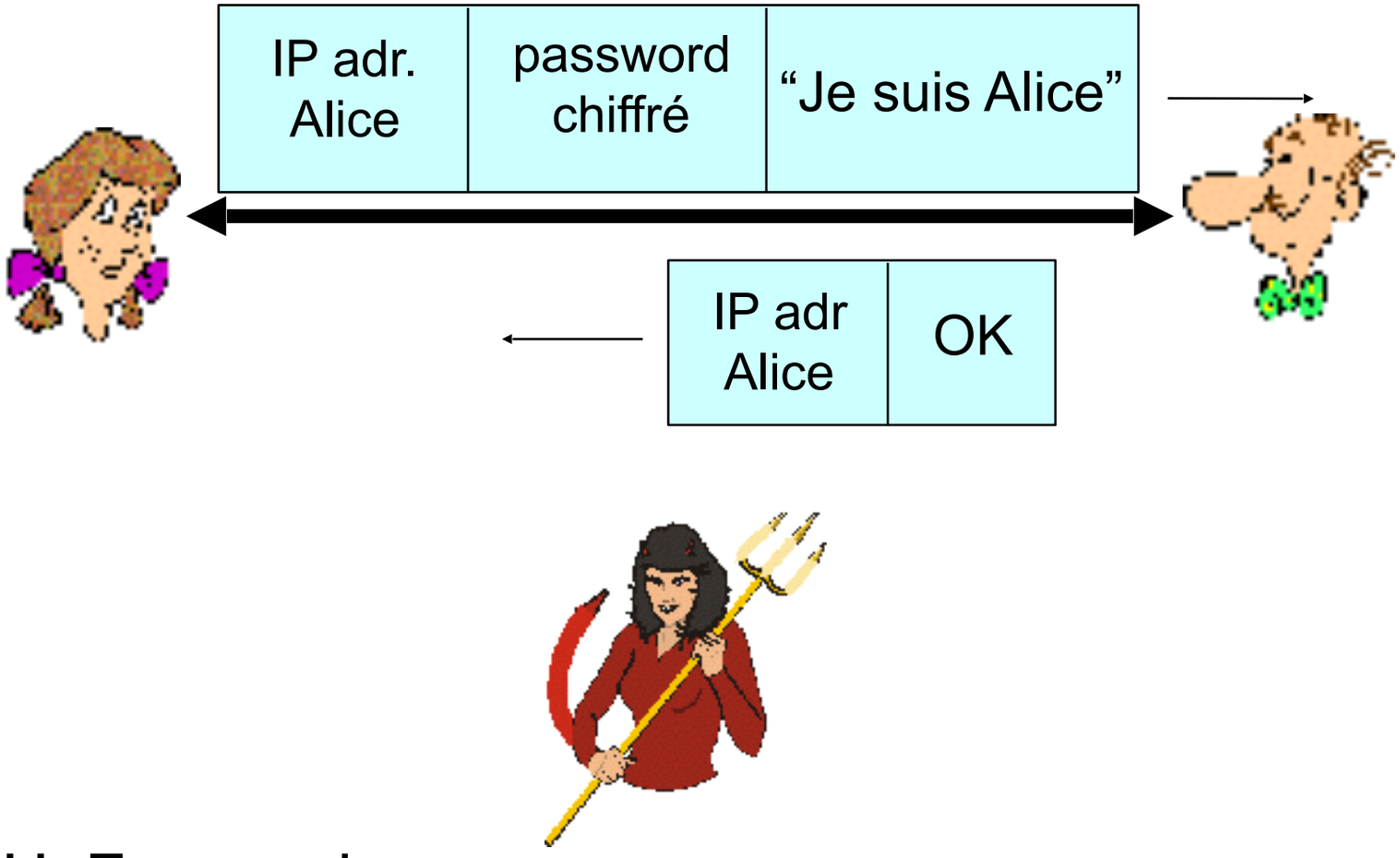
Protocole ap3.0: Alice dit “Je suis Alice” et envoie son password pour le prouver.



attaque par playback: Trudy enregistre le paquet envoyé par Alice et le rejoue vers Bob

Authentification: autre essai

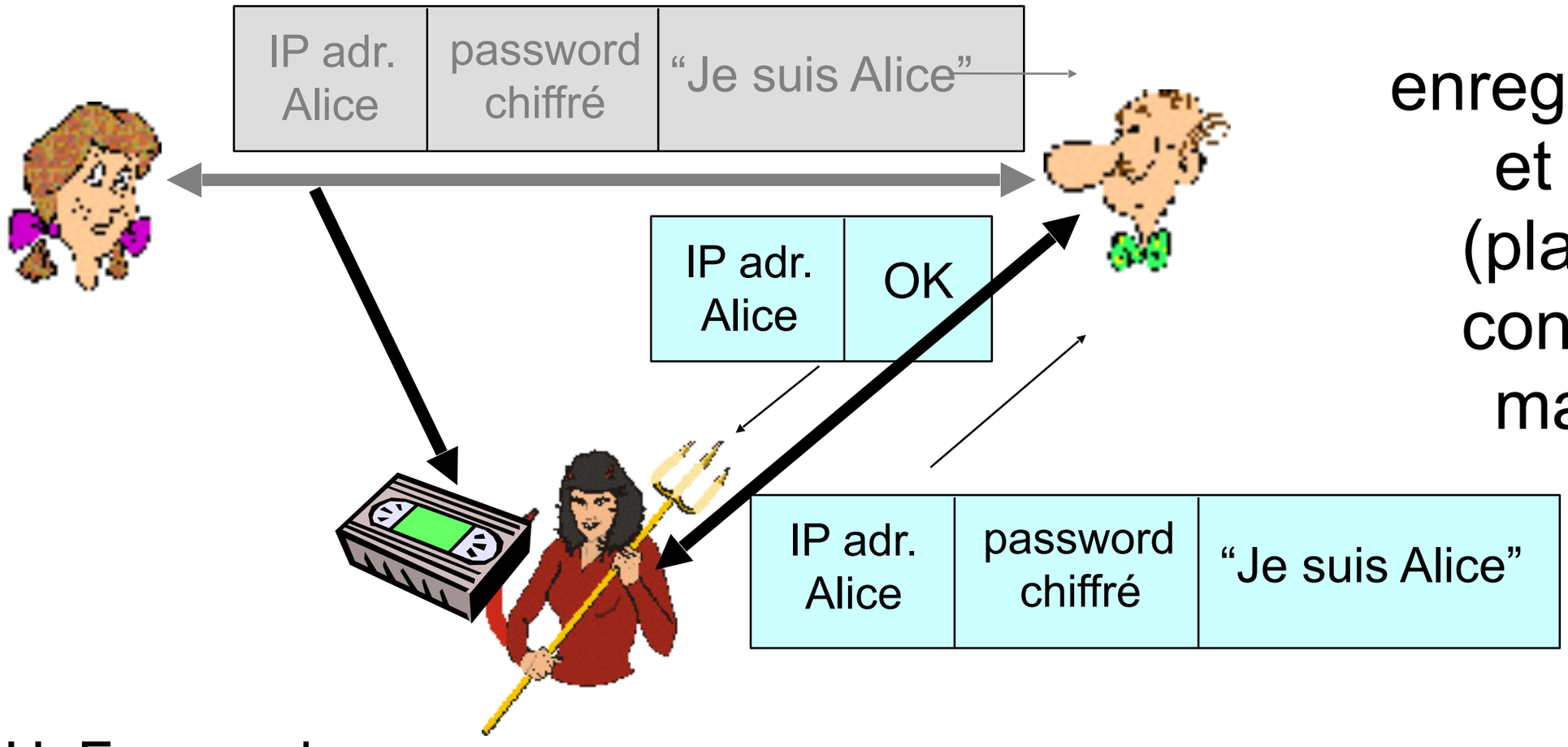
Protocole ap3.01: Alice dit “Je suis Alice” et envoie son password *chiffré* pour le prouver.



La faille??

Authentification: autre essai

Protocole ap3.0: Alice dit “Je suis Alice” et envoie son password chiffré pour le prouver.



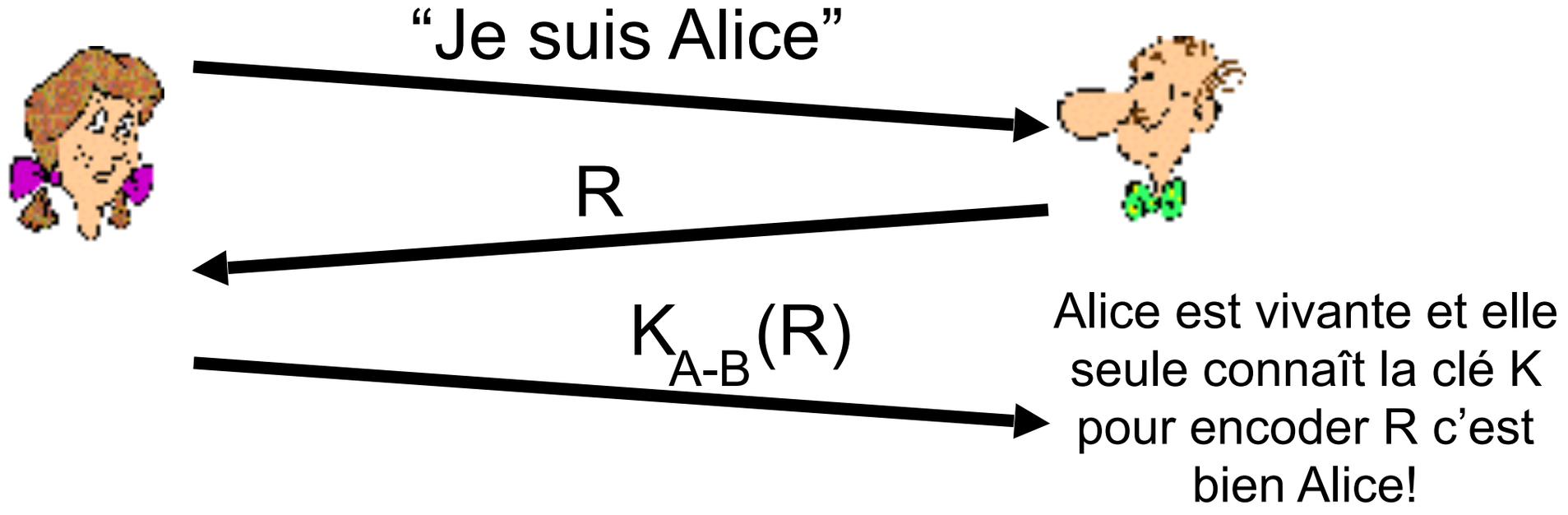
enregistrement et re-jeu (playback) continue à marcher

Authentification: autre essai

But: éviter l'attaque par « playback »

nonce: nombre (R) utilisé (*once-in-a-lifetime)*

ap4.0: pour prouver que c'est la vraie Alice, Bob envoie à Alice un *nonce* R. Alice renvoie R, chiffré avec la clé secrète partagée



Failles, inconvénients?

Authentification:

dans ap4.0 on a utilisé une clé symétrique partagée peut-on utiliser un système à clés publiques?
ap5.0: nonce, + cryptographie à clé publique

Bob calcule

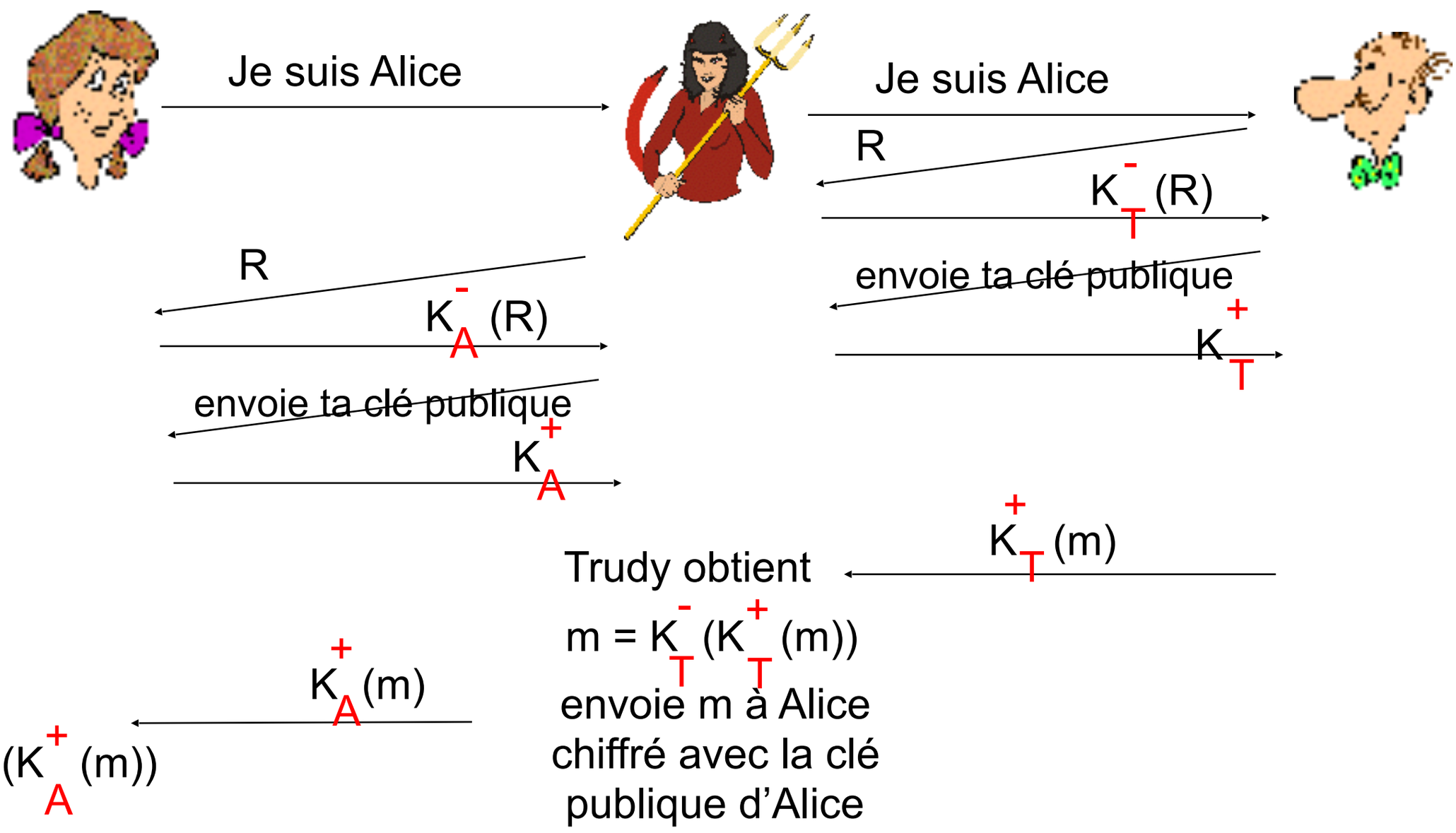
$$K_A^+(K_A^-(R)) = R$$

et sait que seule Alice a la clé privée qui a chiffré R de façon à ce que

$$K_A^+(K_A^-(R)) = R$$

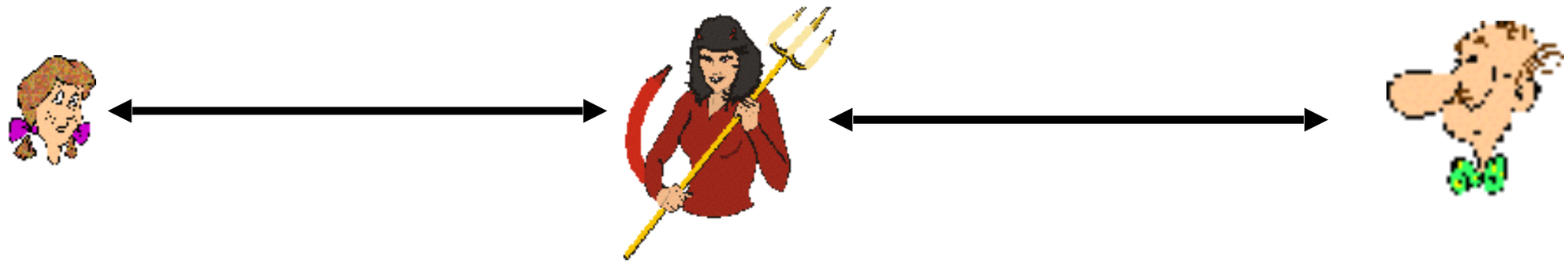
ap5.0: trou de sécurité

man in the middle: Trudy se fait passer pour Alice auprès de Bob et pour Bob auprès d'Alice



ap5.0: trou de sécurité

man in the middle attack: Trudy se fait passer pour Alice auprès de Bob et pour Bob auprès d'Alice



difficile à détecter:

- Bob reçoit tout ce qu'Alice envoie et vice-versa (Bob, Alice peuvent se rencontrer et se rappeler de leur conversation)
- mais Trudy reçoit tous les messages !
- Comment savoir que la clé publique reçue est bien la clé publique d'Alice? (certificats)

Certification des clés publiques

- motivation:
  - Trudy envoie une commande par e-mail: *acheter 4 pizzas*
  - Trudy signe la commande avec sa clé privée
  - Trudy envoie au magasin sa clé publique mais prétend que c'est celle de Bob
  - le magasin vérifie la signature elle envoie les 4 pizzas à Bob
  - Bob n'a rien demandé!

Signatures digitales

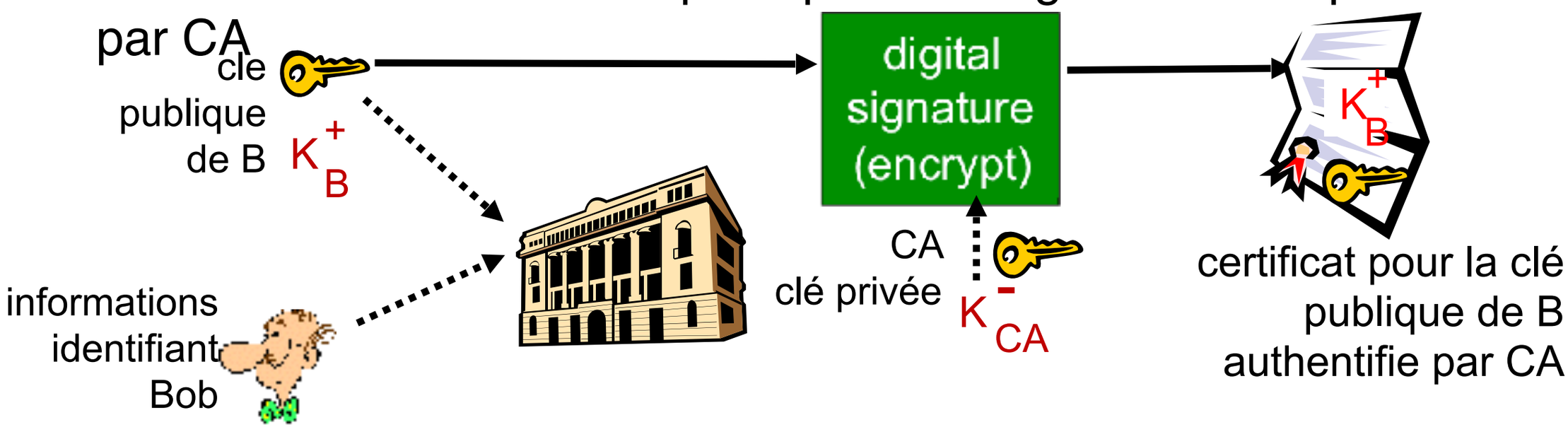
- On suppose un système à clés asymétriques et que seul le possesseur de la clé privée connaît la clé privée et que la clé publique peut être connue de tous.
- Pour signer un document:
  - on signe le document avec la clé privée
  - on envoie le document signé
  - on fournit la clé publique
  - le récepteur vérifie la signature avec la clé publique
- Problème: le récepteur vérifie que le document a été signé par quelqu'un ayant une clé privée correspondant à la clé publique mais cela ne prouve rien…
- il faut certifier la clé publique

Certificat

- Un certificat contient:
  - une clé publique
  - des informations d'identification « certificate subject » (nom, organisation etc..)
  - une signature digitale: le certificat est signé par une entité
  - les informations d'identification de la source du certificat.

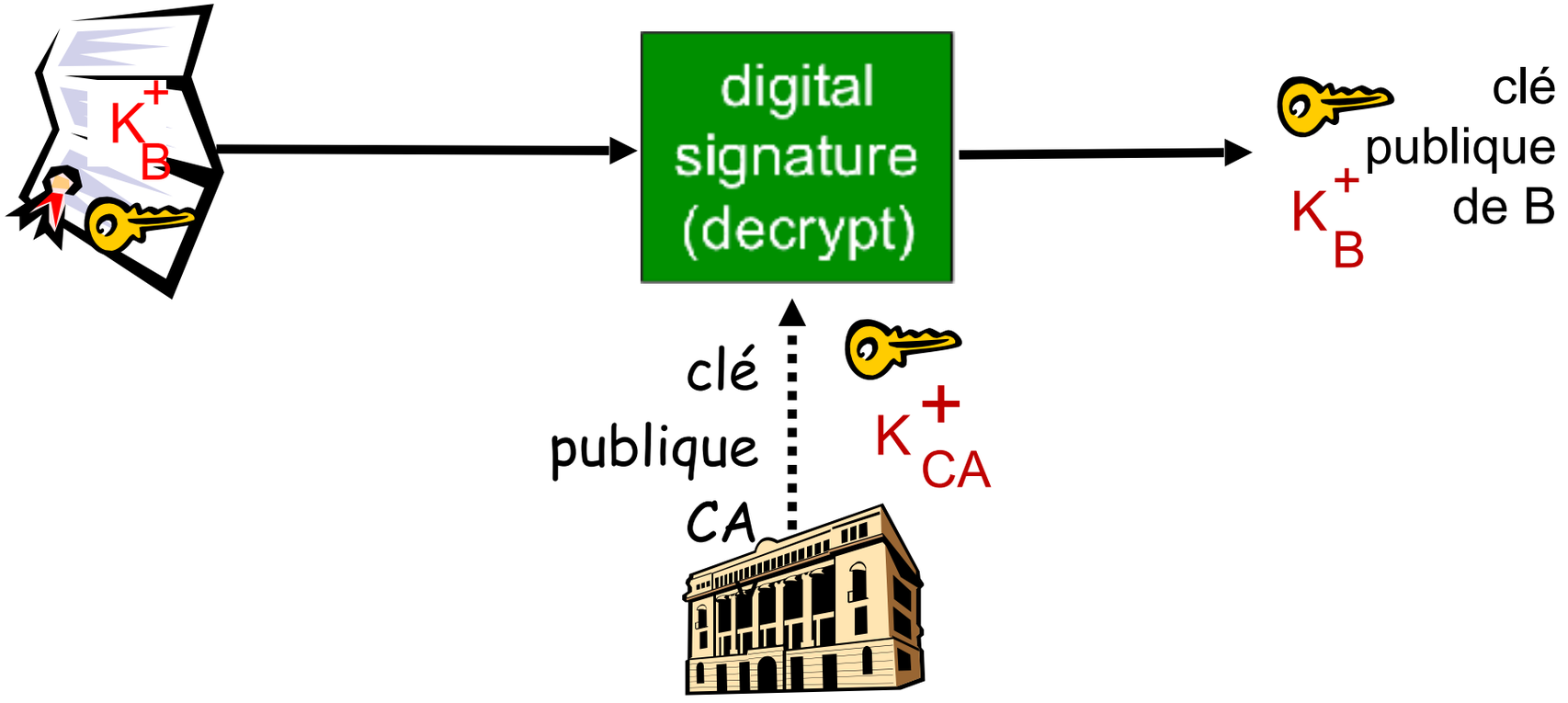
Autorités de certification

- *certification authority (CA):* associe une clé publique à une entité E.
- E (personne, site) enregistre sa clé publique auprès de l'autorité de certification
- E fournit la preuve de l'identité par la CA.
  - CA crée un certificat associant E à sa clé publique.
  - ce certificat contient la clé publique de E signée numériquement par CA



Autorités de certification

- quand Alice veut obtenir la clé publique de Bob:
  - elle obtient le certificat de Bob (de n'importe qui).
  - applique la clé publique de CA au certificat de Bob, et obtient la clé publique de Bob.



Vérifier le certificat

- Pour vérifier le certificat:
  - la source (*issuer*) est une autorité « connue » avec une clé publique connue qui permet d'authentifier le certificat
  - sinon il faut vérifier la clé publique du certificat avec un autre certificat..
  - jusqu'à obtenir une clé publique en qui on a confiance
  - on a ainsi une chaine de certificats
- Mais… ce n'est pas toujours possible: on peut générer une empreinte (fingerprint) du certificat et vérifier physiquement auprès de la source que cette empreinte est la bonne

Autorités de certifications

- *Certification authority* (CA) certifie des clés publiques:
  - requête: self-signed certificate (CSR) vers une autorité de certification,
  - CA vérifie votre identité (avec des moyens autres) et établit un certificat pour votre clé publique signée avec la clé privée du CA.
  - (éventuellement une chaine de certificats)

Keystores

- Les certificats de confiance sont stockés dans le « keystore » comme étant des « trusted certificates »
- les clés publiques de ces certificats peuvent être utilisées pour vérifier les signatures
- Pour envoyer un code ou un document signé il faut joindre le certificat qui certifie la clé publique correspondant à la clé privée utilisée dans la signature
- les « keystores » contiennent:
  - les certificats de confiance
  - les couples clés privée / certificat de la clé

keytool

- keytool est un outil java pour:
  - créer des clés privées/ publiques
  - créer des requêtes de certificat vers des CA
  - importer des réponses à ces requêtes provenant de CA
  - importer des certificats de clés publiques
  - gérer le keystore

Exécuter du code…

- exécuter un code dont on ne connaît pas la provenance est extrêmement dangereux:
  - soit limiter les possibilités de ce code (security manager)
  - soit avoir la garantie de sa provenance: signatures et authentifications

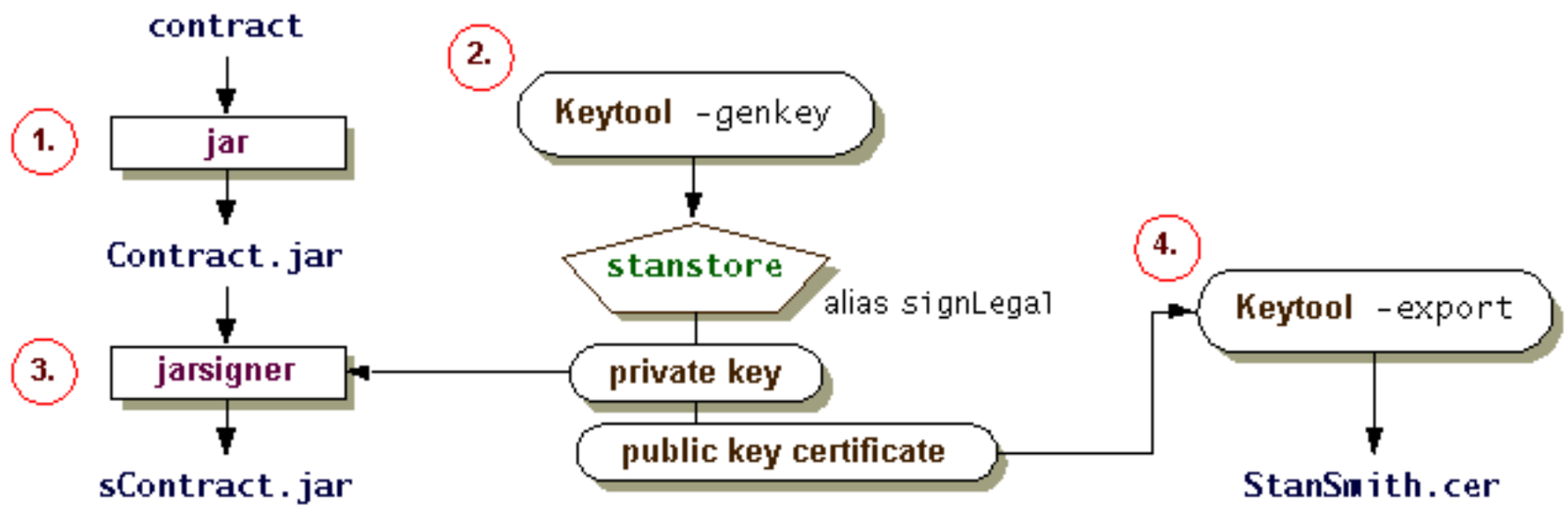
Comment faire?

- en utilisant JAR et (jarsigner)
  - créer un fichier JAR
  - générer des clés (keytool -genkey)
    - avec keytool -certreq faire une requête de certificat auprès d'une CA
    - avec keytool -import importer la réponse
  - signer le fichier JAR avec jarsigner et la clé privée
  - exporter le certificat de la clé publique: keytool -export

Avec les autorités

- keytool génère un certificat auto-signé (signé avec la clé privée)
- Pour avoir un certificat signé par une autorité de certification:
  - générer un Certificat Signing Request (CSR)
    - *keytool -certreq -alias unAlias -file csrFile*
  - soumettre *csrFile* à une autorité de certification
  - qui retourne un certificat signé (ou une chaîne) par le CA
    - *keytool -import -alias alias -file ABCCA.cer -keystore storefile* ajoute le certificat
  - *keytool -import -trustcacerts -keystore storefile -alias alias*
  - *-file certReplyFile* insère la réponse au CSR (avec vérification)

Envoyer un contrat de StanSmith



- Créer le JAR (jar cvf Contract.jar contract)
- générer les clés
- signer le JAR
- exporter le certificat

Générer les clés

```
hf$ keytool -genkey -alias signLegal -keystore examplestore
Entrez le mot de passe du fichier de clés :
Le mot de passe du fichier de clés est trop court : il doit comporter au moins 6 caractères
Entrez le mot de passe du fichier de clés :
Ressaisissez le nouveau mot de passe :
Quels sont vos nom et prénom ?
  [Unknown]:  Stan Smith
Quel est le nom de votre unité organisationnelle ?
  [Unknown]:  Legal
Quel est le nom de votre entreprise ?
  [Unknown]:  UFRInfo
Quel est le nom de votre ville de résidence ?
  [Unknown]:  Paris
Quel est le nom de votre état ou province ?
  [Unknown]:  PARIS
Quel est le code pays à deux lettres pour cette unité ?
  [Unknown]:  FR
Est-ce CN=Stan Smith, OU=Legal, O=UFRInfo, L=Paris, ST=PARIS, C=FR ?
  [non]:  oui

Entrez le mot de passe de la clé pour <signLegal>
   (appuyez sur Entrée s'il s'agit du mot de passe du fichier de clés) :

Un keystore exmplestore a été créé avec clé privée clé publique pour Stan Smith avec un
certificat auto-signé  (il est valide 90 jours) et est associé à la clé privée identifiée
comme signLegal
```

Signer le JAR

```
hf$ jarsigner -keystore examplestore -signedjar sContract.jar Contract.jar signLegal
Enter Passphrase for keystore:
jar signed.

Warning:
The signer certificate will expire within six months.
No -tsa or -tsacert is provided and this jar is not timestamped. Without a
timestamp, users may not be able to validate this jar after the signer certificate's
expiration date (2015-06-30) or after any future revocation date.
```

Exporter le certificat

```
hf$ keytool -export -keystore examplestore -alias signLegal -file StanSmith.cer
Entrez le mot de passe du fichier de clés :
Certificat stocké dans le fichier <StanSmith.cer>
```

Récepteur…

- Ruth a reçu de StanSmith
  - sContract.jar le JAR signé
  - le certificat *StanSmith.cer* (qui contient la clé privée et la clé publique)
  - il faut importer le certificat dans le keystore:

Récepteur…

```
hf$ keytool -import -alias -stan -file StanSmith.cer -keystore exempleruth
Entrez le mot de passe du fichier de clés :
Ressaisissez le nouveau mot de passe :
Propriétaire : CN=Stan Smith, OU=Legal, O=UFRInfo, L=Paris, ST=PARIS, C=FR
Emetteur : CN=Stan Smith, OU=Legal, O=UFRInfo, L=Paris, ST=PARIS, C=FR
Numéro de série : f0dce85
Valide du : Wed Apr 01 17:47:39 CEST 2015 au : Tue Jun 30 17:47:39 CEST 2015
Empreintes du certificat :
    MD5:  2B:AC:09:AB:AE:D7:A0:76:68:16:13:22:AC:BF:D9:87
    SHA1 : 8E:3B:C3:24:C5:23:02:0E:B2:4A:BA:41:3A:2E:E7:79:66:BF:89:1D
    SHA256 : B0:22:7B:F8:3E:21:A3:F9:A2:02:F5:59:80:A8:92:FB:5C:FE:A1:AF:1F:1F:
28:A7:41:9E:EE:F5:D4:9D:15:AE
    Nom de l'algorithme de signature : SHA1withDSA
    Version : 3

Extensions :

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B3 53 53 0F F2 E6 30 90   FD B8 2E 25 1E 82 72 B3  .SS...0....%..r.
0010: 79 05 87 C6                                        y...
]
]

Faire confiance à ce certificat ? [non] :  oui
Certificat ajouté au fichier de clés
```

Vérifier le certificat

- Ruth téléphone à Stan. Stan a le « fingerprint » du certificat:

```
 hf$ keytool -printcert -file StanSmith.cer
Propriétaire : CN=Stan Smith, OU=Legal, O=UFRInfo, L=Paris, ST=PARIS, C=FR
Emetteur : CN=Stan Smith, OU=Legal, O=UFRInfo, L=Paris, ST=PARIS, C=FR
Numéro de série : f0dce85
Valide du : Wed Apr 01 17:47:39 CEST 2015 au : Tue Jun 30 17:47:39 CEST 2015
Empreintes du certificat :
    MD5:  2B:AC:09:AB:AE:D7:A0:76:68:16:13:22:AC:BF:D9:87
    SHA1 : 8E:3B:C3:24:C5:23:02:0E:B2:4A:BA:41:3A:2E:E7:79:66:BF:89:1D
    SHA256 : B0:22:7B:F8:3E:21:A3:F9:A2:02:F5:59:80:A8:92:FB:5C:FE:A1:AF:1F:1F:
28:A7:41:9E:EE:F5:D4:9D:15:AE
    Nom de l'algorithme de signature : SHA1withDSA
    Version : 3

Extensions :

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B3 53 53 0F F2 E6 30 90   FD B8 2E 25 1E 82 72 B3  .SS...0....%..r.
0010: 79 05 87 C6                                        y...
]
]
```

Vérifier la signature..

- Ruth:

```
hf$ jarsigner -verify -verbose -keystore exempleruth sContract.jar

s k      148 Wed Apr 01 17:58:38 CEST 2015 META-INF/MANIFEST.MF
         310 Wed Apr 01 17:58:38 CEST 2015 META-INF/SIGNLEGA.SF
        1057 Wed Apr 01 17:58:38 CEST 2015 META-INF/SIGNLEGA.DSA
           0 Wed Apr 01 17:42:00 CEST 2015 META-INF/
smk       15 Wed Apr 01 17:41:42 CEST 2015 contract

  s = signature was verified
  m = entry is listed in manifest
  k = at least one certificate was found in keystore
  i = at least one certificate was found in identity scope

jar verified.

Warning:
This jar contains entries whose signer certificate will expire within six months.
This jar contains signatures that does not include a timestamp. Without a timestamp,
users may not be able to validate this jar after the signer certificate's expiration date
(2015-06-30) or after any future revocation date.

Re-run with the -verbose and -certs options for more details.
```

Résultat:

```
hf $jar xvf sContract.jar
décompressé : META-INF/MANIFEST.MF
décompressé : META-INF/SIGNLEGA.SF
décompressé : META-INF/SIGNLEGA.DSA
 créé : META-INF/
décompressé : contract
hf$ more contract
Je m'engage...
```

Et maintenant avec du code…

```
import java.io.*;
public class Compter {
    public static void countChars(InputStream in) throws IOException
    {
        int cmp = 0;
        while (in.read() != -1)
            cmp++;
        System.out.println("On a " + cmp + " caractères.");
    }
    public static void main(String[] args) throws Exception
    {
        if (args.length >= 1)
            countChars(new FileInputStream(args[0]));
        else
            System.err.println("Usage: Compte fichier");
    }
}
```

```
java Compter.java
jar cvf Compter.jar Compter.class
```

keytool

```
hf$ keytool -genkey -alias signFiles -keystore exemplestore
Entrez le mot de passe du fichier de clés :
Ressaisissez le nouveau mot de passe :
Quels sont vos nom et prénom ?
  [Unknown]:  Hugues Fauconnier
Quel est le nom de votre unité organisationnelle ?
  [Unknown]:  LIAFA
Quel est le nom de votre entreprise ?
  [Unknown]:  paris-diderot
Quel est le nom de votre ville de résidence ?
  [Unknown]:  PARIS
Quel est le nom de votre état ou province ?
  [Unknown]:  FR
Quel est le code pays à deux lettres pour cette unité ?
  [Unknown]:  FR
Est-ce CN=Hugues Fauconnier, OU=LIAFA, O=paris-diderot, L=PARIS, ST=FR, C=FR ?
  [non]:  oui

Entrez le mot de passe de la clé pour <signFiles>
   (appuyez sur Entrée s'il s'agit du mot de passe du fichier de clés) :
Ressaisissez le nouveau mot de passe :

(un alias signFiles pour Hugues Fauconnier a été ajouté dans le keystore
exemplestore)
```

Signer le JAR

```
jarsigner -keystore exemplestore -signedjar sCompter.jar Compter.jar
signFiles
Enter Passphrase for keystore:
jar signed.

Warning:
The signer certificate will expire within six months.
No -tsa or -tsacert is provided and this jar is not timestamped. Without
a timestamp, users may not be able to validate this jar after the signer
certificate's expiration date (2015-06-30) or after any future
revocation date.
```

le jar est signé par l'alias signFiles du keystrore exemplestore

pour exporter le certificat:

```
hf$ keytool -export -keystore exemplestore -alias signFiles -file
Exemple.cer
Entrez le mot de passe du fichier de clés :
Certificat stocké dans le fichier <Exemple.cer>
```

un certifcicat exportable dans Exemple.cer

Du côté du récepteur…

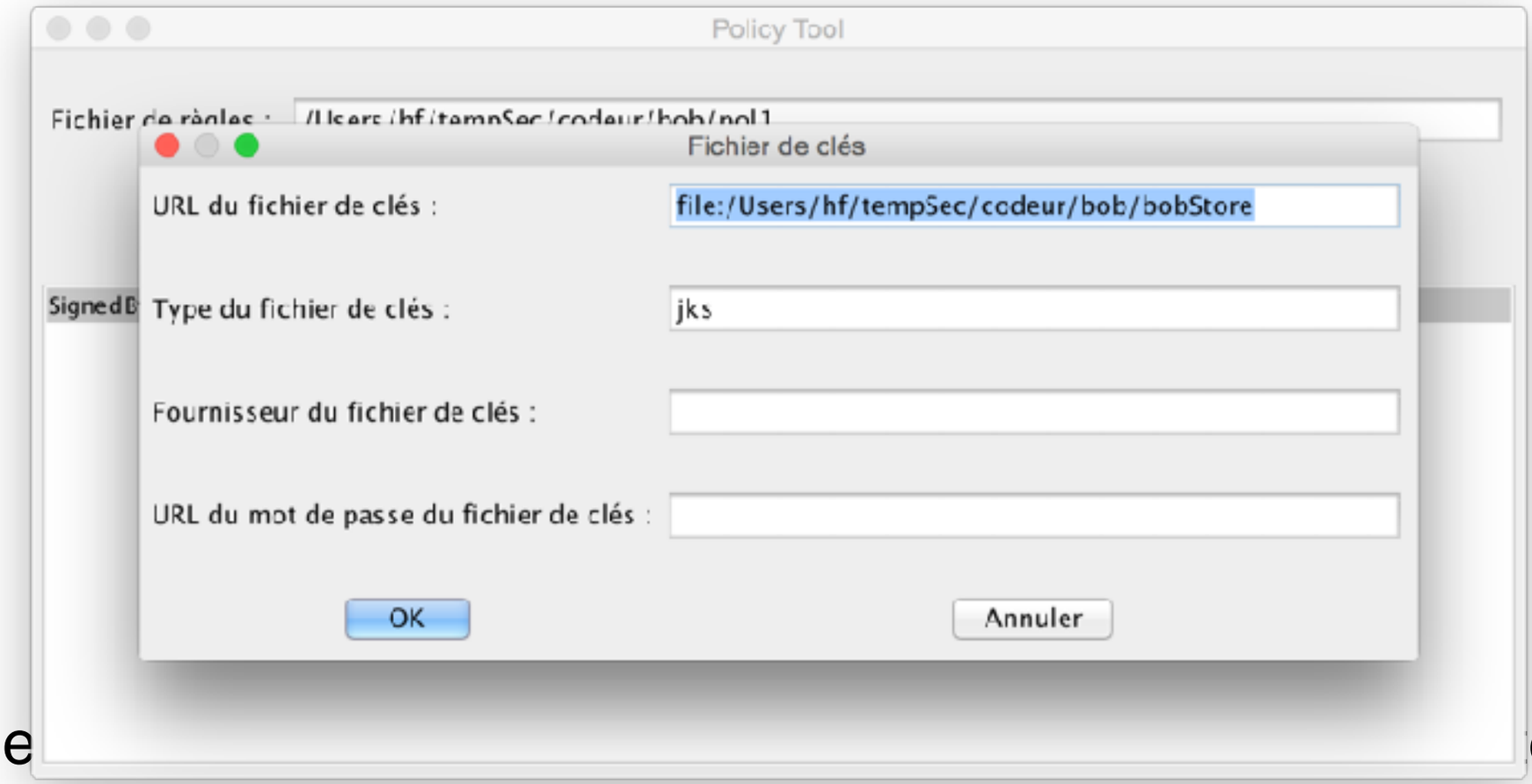
- Bob importe le certificat: (et le vérifie auprès de Hugues)

```
hf$ keytool -import -alias bob -file Exemple.cer -keystore bobStore
Entrez le mot de passe du fichier de clés :
Ressaisissez le nouveau mot de passe :
Propriétaire : CN=Hugues Fauconnier, OU=LIAFA, O=paris-diderot, L=PARIS, ST=FR, C=FR
Emetteur : CN=Hugues Fauconnier, OU=LIAFA, O=paris-diderot, L=PARIS, ST=FR, C=FR
Numéro de série : 7136d79f
Valide du : Wed Apr 01 18:35:15 CEST 2015 au : Tue Jun 30 18:35:15 CEST 2015
Empreintes du certificat :
    MD5:  DD:2C:63:A0:8C:41:EC:8B:93:C8:3B:63:FA:30:C0:1D
    SHA1 : CC:D3:DE:04:63:F7:CE:AD:7D:3B:BC:2E:5D:C7:6D:B6:82:79:66:D6
    SHA256 : CD:3A:A9:01:AD:CE:35:C3:87:22:B0:30:AA:34:06:C9:CF:DA:EB:C0:F4:70:B4:1C:
13:39:1C:CE:A2:75:50:55
    Nom de l'algorithme de signature : SHA1withDSA
    Version : 3

Extensions :

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 14 29 A9 EF 27 B1 CF 18   35 C3 AC A2 E2 FD C6 98  .)..'...5.......
0010: D6 37 5C CE                                        .7\.
]
]

Faire confiance à ce certificat ? [non] :  oui
Certificat ajouté au fichier de clés
```

dans le keystore de bob (bobStore) bob est un alias pour le nouveau certificat Exemple.cer

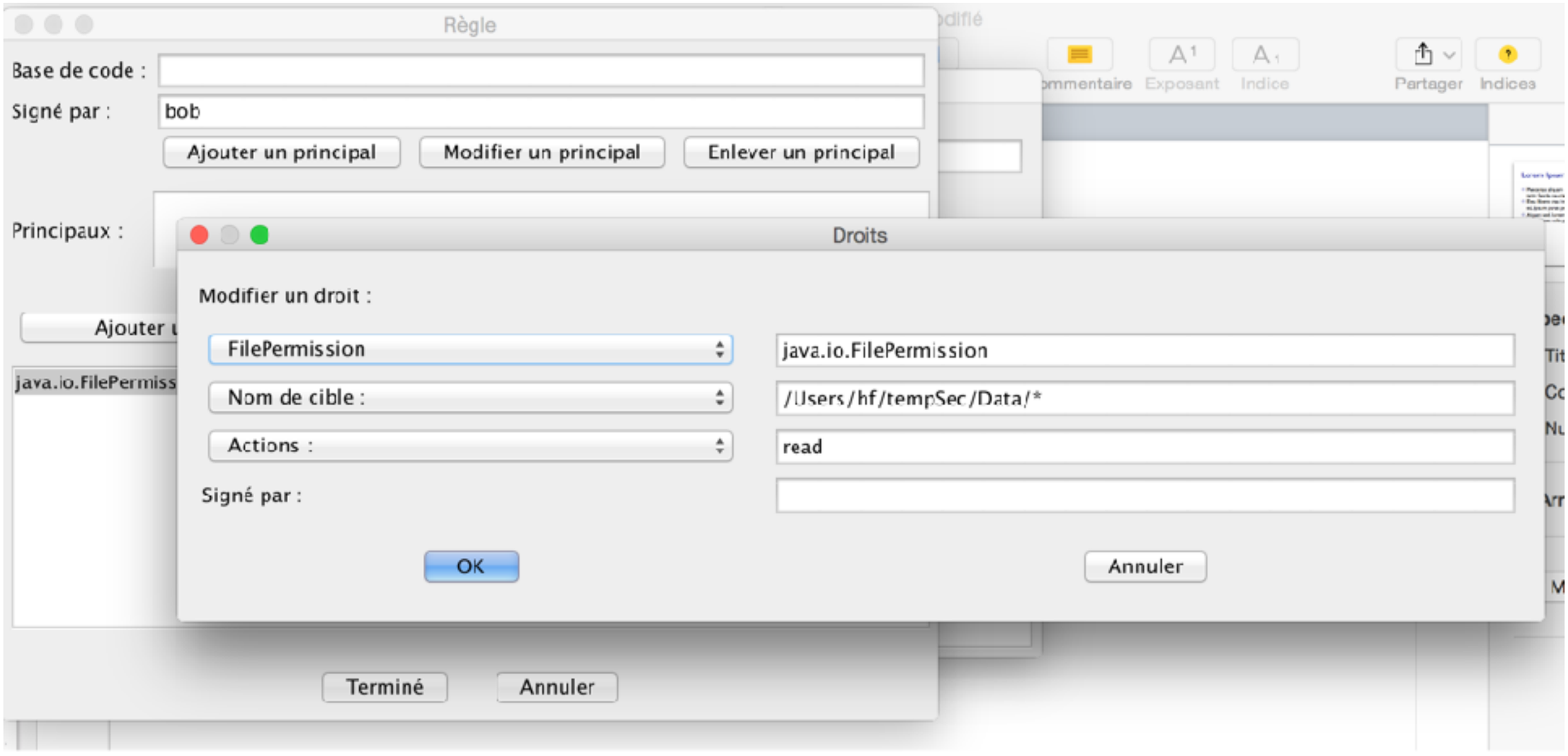
Côté du récepteur: security

```
java -cp sCompter.jar Compter /Users/hf/tempSec/Data/Fichier
On a 491 caractères
: ok pas de security manager

java -Djava.security.manager -cp sCompter.jar Compter /Users/hf/Data/Fichier :
exception droit en lecture

policytool&

définir l'url du keystore:
```



Côté du récepteur

- keytool: donner les droits en lecture



Côté récepteur

sauvegarde de la politique de sécurité dans le fichier pol1:

```
/* AUTOMATICALLY GENERATED ON Wed Apr 01 19:19:19 CEST 2015*/
/* DO NOT EDIT */

keystore "file:/Users/hf/tempSec/codeur/bob/bobStore", "jks";

grant signedBy "bob" {
  permission java.io.FilePermission "/Users/hf/tempSec/Data/*", "read";
};
```

```
java -Djava.security.manager -Djava.security.policy=pol1 -
cp sCompter.jar Compter /Users/hf/tempSec/Data/Fichier
```

On a 491 caractères