JavaCard

Anas Abou El Kalam
anas.abouelkalam@enseeiht.fr

Agenda

- Les Cartes à Puces
- La JavaCard
- Sécurité JavaCard

Agenda

Les Cartes à Puces
- Introd
- Fonctionalités
- Capacités
- Applis
- Technos
- Communication avec la carte
- L'APDU
- Les cartes à puces & la sécurité

La JavaCard

Sécurité JavaCard

Intro

La carte à puce
- Support électronique permettant d'intégrer applis à haut niveau d'intégrité et de confidentialité dans des systèmes d'information.
- Offre espace sécurisé pour des applis «sensibles»
- Séparation entre sécurité du système et celle des applications
- En résumé : objet portable, personalisé et sécurisé
  - ... devenue *plate forme d'exécution à part entière*

JavaCard (Techno java)
- Fournit plate-forme d'exécution d'applis utilisant technos objet
- Améliorer interopérabilité
- Permet d'intégrer plusieurs applications
- Cartes deviennent support générique permettant d'embarquer ++ applis
- Développeur n'ait plus à se préoccuper des protocoles de com entre carte et son environnement, en déléguant génération trames de com à outils dédiés
  - Sécurité : matérielle + logicielle // fautes conception, malveillances, ...

Fonctionnalités carte à puce

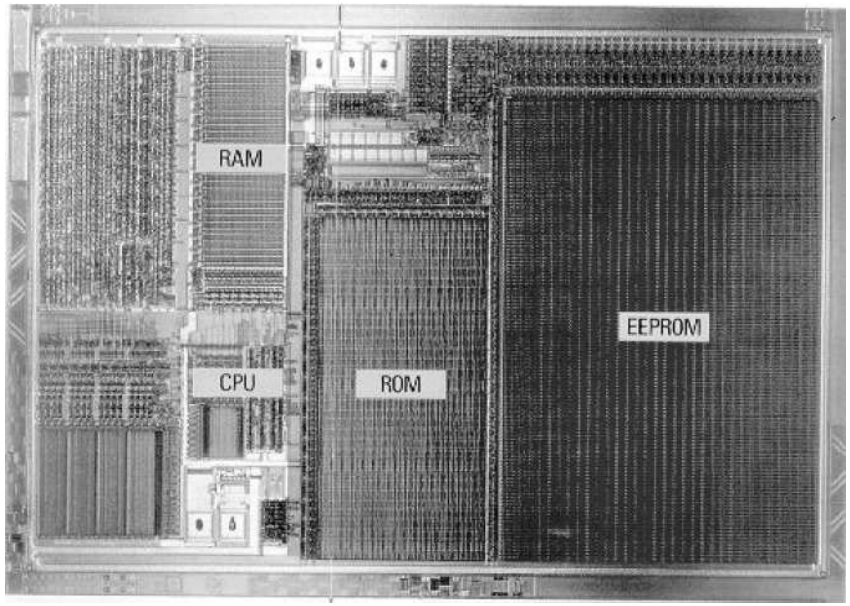
La carte :

- support électronique mobile et fiable
- environnement de stockage & calcul sécurisé
  - Conserver secrets
  - Exécuter prog critiques
  - Chiffrement / Scellement / signature : RSA, AES, MD5, ...
    - ==> Seul le détenteur de la carte peut coder / décoder / exécuter ...

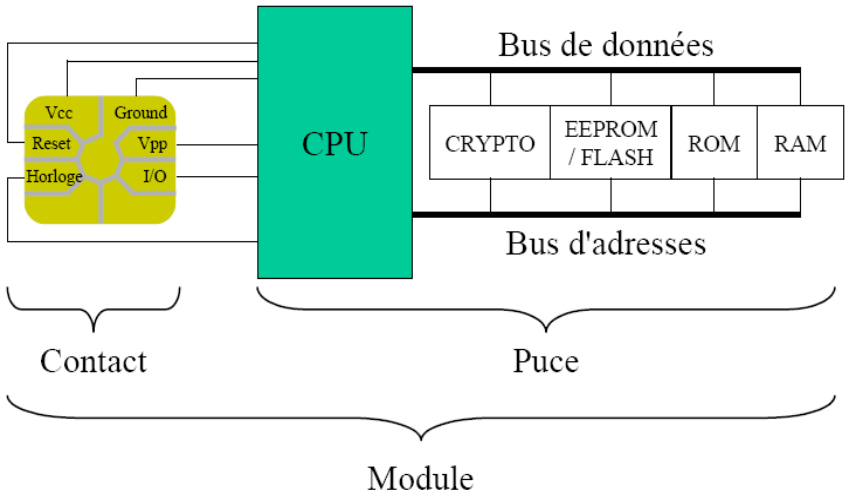
Caractéristiques // Avantages ...

- Taille permet d'affronter les tests de torsion sans être détériorée
- Microprocesseur est un bloc monolithique qui contient
  - Unités de calcul tels le processeur et le co-processeur crypto,
  - Mémoires contenant le code de la carte dans la ROM,
  - Mémoires de travail, persistantes comme l'EEPROM,
  - Mémoire temporaires telle la RAM
  - Eléments de communication

Architecture : carte à microprocesseur



Architecture (2)



Capacités carte à puce

Année	Taille du bus	Fréquence	RAM	Mémoire persistante
1981	8 bits	4,77 MHz	36 octets	1 ko
1985	8 bits	4,77 MHz	128 octets	2 ko
1990	8-16 bits	4,77 MHz	256 octets	8 ko
1996	8-32 bits	4,77-28,16 MHz	512 octets	32 ko
2000	8-32 bits	4,77-28,16 MHz	1536 octets	64 ko
2002	8-32 bits	4,77-28,16 MHz	3 à 4 ko	128 ko

On peut espèrer une constante croissance des capacités des cartes grâce à

- la gravure de plus en plus fine des puces
- l'apparition des nouvelles technos mémoires comme la FeRAM (Ferroelectric RAM)
  - Mémoire Volatille, rapide, nécessite moins de tension, ...

Carte à puce : les applis

Télécarte

- Carte à mémoire contenant compteur
  - Ce compteur représente unités dispos sur carte et est décrémenté lors de l'utilisation dans une cabine publique

Cartes bancaire

réduire fraudes (fausse monnaie, chèque non endossable) et faciliter transactions

porte monnaie électronique (e.g., Moneo)

- pré-charger électroniquement le porte monnaie

Cartes à microprocesseur

- capables d'exécuter des applications
- Atout : personnalisation, inviolabilité
- e.g., carte d'identité, reconnaissance biométrique, ...

Carte à puce : Ex d'applis

Télécarte

- Carte à mémoire contenant compteur
  - Ce compteur représente unités dispos sur carte et est décrémenté lors de l'utilisation dans une cabine publique

Cartes bancaire

réduire fraudes (fausse monnaie, chèque non endossable) et faciliter transactions

porte monnaie électronique (e.g., Moneo)

- pré-charger électroniquement le porte monnaie

Cartes à microprocesseur

- capables d'exécuter des applications
- Atout : personnalisation, inviolabilité
- e.g., carte d'identité, reconnaissance biométrique, cartes SIM,

Cartes multi-applicatives

- Applis indépendantes ou partageant des infos / services ...

Carte à puce : Technologies

2 catégories selon architecture matérielle :

cartes à mémoire et cartes à microprocesseurs

2 catégories selon technologie utilisée pour communiquer avec l'extérieur :

- cartes à contact et cartes sans-contact

Cartes à mémoire

- réunissent sur même circuit intégré
  - *mémoire non volatile* : stocker quelques centaines de bits
  - un peu de logique : read/ write exécuter des algos d'auth rudimentaires
    - compteur unités, N° série, donnée secrète pour authentifier carte

Cartes à micropocesseur

un véritable système sur composant intégrant tous les éléments matériels et ogiciels d'un ordinateur sur un seul composant.

- microcontrôleur
  - microprocesseur, mémoire morte, mémoire vive, périphs pour effectuer opérations spécialisées (gestion ligne série, opérations arithmétiques sur grands nombres, génération nombres aléatoires ...)

Carte à puce : Paltes formes ouvertes

Depuis quelques années, les *plate-formes ouvertes* ont fait leur apparition

- JavaCard (promu par Sun)
- Multos (utilisé pour l'application Mondex)
- Windows for Smart Card (de Microsoft)

Ces plate-formes ne contiennent que le SE, avec une

- couche de chargement d'applications:
- Les applications sont développées dans un langage intermédiaire
  - JavaCard = Java
  - Multos = C
  - WfSC = Visual Basic
- Puis compilées dans un format de chargement.
- JavaCard et Multos ont un système de chargement d'application d'une grande sécurité

Carte à puce : Manufacturiers

Manufacturiers de cartes

- Gemplus
- Oberthur
- Schlumberger
- Orga
- De La Rue
- Bull

Manufacturiers de puce

- Infineon (Siemens)
- NEC
- Philips
- SGS-Thomson
- Hitachi
- Texas Instruments

Carte à puce : Normes OSI

7816-1: Format de la carte

7816-2: Position des contacts

7816-3: Réponse au démarrage (ATR) et protocoles de communication (T=0, T=1)

7816-4: Commandes inter-industrie

7816-5: Système d'enregistrement des applications

7816-6: Éléments de données inter-industrie

7816-7: Commandes inter-industrie pour SCQL

7816-8: Architecture de sécurité et commandes interindustrie pertinentes

Comm entre Carte & monde extérieur

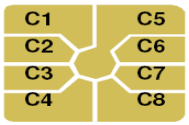
Les cartes disposent de huit contacts dont l'utilisation est standardisée par la norme ISO 7816-2

La com entre carte et extérieur se fait par l'intermédiaire d'une ligne série disponible sur le contact C7

Protocole USB (de type série) même si données transmises sont codées de façon différentielle sur les deux contacts C4 et C8

norme ISO 7816-3 définit : T=0 & T=1

- Protocoles de com pour transporter des données sur ligne d'E/S
- protocoles asynchrones, semi-duplex et transportent des blocs de données appelés APDU (Application Protocol Data Unit)



Contact	Utilisation
C1	Tension d'alimentation
C2	Remise à zéro
C3	Horloge
C4	USB +
C5	Masse
C6	Tension de programmation (obsolète)
C7	Données
C8	USB −

Comm entre Carte & monde extérieur

La communication avec une carte se fait via un lecteur.

Lorsque la carte est mise sous tension, elle émet une ***réponse au démarrage (Anwer-To-Reset)***

Cette réponse indique au lecteur les capacités de la carte à puce concernant les protocoles de communications:

- Type de protocole (T=0, T=1, ...)
- Vitesse de transmission (9600 bauds, 38400 bauds)
- Extension du délai permis d'exécution d'une commande

L'ATR contient aussi des informations dites historiques, dont le format est aussi normalisé par ISO

- On retrouve dans ces informations l'identification du produit, sa version, le type de puce utilisé, l'état de la carte

Comm entre Carte & monde extérieur

Exemple d'une ATR

- TS : 3Bh Caractère de synchronisation
- T0 : 19h Paramètre TA1 présent; 9 octets historiques
- TA1 : 95h Vitesse de protocole : 38400 bauds
- T1 : 64h 5 octets d'information pré-émission
- T2 : 40h Famille de produit
- T3 : 42h Identification du SE
- T4 : 11h Version du SE 1.1
- T5 : CCh Identification de la puce
- T6 : 83h 3 octets d'information de statut
- T7 : 1Fh État de la carte
- T8 : 90h SW1 au démarrage
- T9 : 00h SW2 au démarrage

L'APDU

Pile de protocoles de communication extrêmement réduite :

- T=0 et T=1 : couche 2 du modèle OSI
- APDU : couche 7
- Le niveau 1 est également spécifié dans la norme ISO 7816-3

2 catégàries d'APDU

- **commandes**
  - CLA : domaine d'appli (e.g., commandes de carte SIM utilisent la classe A0h)
  - INS : identifie requête qui est faite à la carte
  - P1 ou P2 : octets de paramètres
  - Lc / Le : longueur des données de la Commande / Réponse
- **Réponses**
  - SW1 et SW2 : code indiquant si commande s'est correctement déroulée

APDU de commande :

APDU de réponse:

L'APDU : familles de commandes

Gestion des fichiers

- Sélectionner, Lire, Écrire, Créer, Détruire

Sécurité

- Vérifier un NIP, Changer un NIP, Déverrouiller un NIP
- Authentification mutuelle, Établir une clé de session

Cryptographique

- Signer un bloc de données, Vérifier la signature, Générer une clé

Paiement

- Débit, Crédit

Personnalisation

- Lire la mémoire, Écrire dans la mémoire

Les cartes à puces et la sécurité

Gestion intelligente des NIPS

- Compteur de présentation et verrouillage du NIP après un nombre donné d'essais ratés

Conservation sécuritaire des NIPs et des clés

- Difficile d'attaquer physiquement les composantes
- Attaques détournées (délais, analyse de la puissance utilisée, analyse différentielle des fautes)

Génération et conservation des clés privées dans la carte

- Un calcul cryptographique, comme la signature, est alors fait par la carte, sans que la clé ait à être transmise au PC
- Mêmes types d'attaque que pour les NIPs

Les cartes à puces et la sécurité

Gestion du cycle de vie de la carte grâce à un indicateur en OTP

- Puce => Carte => Personnalisation => Application

Architecture particulière et contre-mesures proposées par le manufacturier de puces

- Brouillage du bus
- Chiffrement du bus et des données
- Disposition des éléments physiques
- Grille de détection d'intrusion
- Détecteurs de conditions anormales (tension, fréquence externe, matrice de contrôle d'accès à la mémoire, numéro de série unique par puce)

Agenda

Les Cartes à Puces

La JavaCard

- Le langage / API
- Squelle d'application Javacard
- Architecture
- JCVM
- Packetages & Extensions

Sécurité JavaCard

Les Javacard

Quoi ?

- Adaptation / réduction de spéc. Java pour tenir compte des spécificités de carte

Pourquoi ?

- limiter nombre produits ≠ à développer pour satisfaire besoins ++ secteurs
- séparer applis du SE et de l'env d'exécution, autorisant la diffusion de cartes à puce génériques et d'applets portables
- Portabilité code entre les différentes cartes disponibles
- chargement d'application dynamiquement durant la vie de la carte
- présence de ++ applis sur même carte pouvant être sélectionnées successivement
- structurer des applications librement en gérant des objets persistants, sans forcément passer par un système de fichier.

Comment ?

- Garder principes de base du langage Java (orienté objet)
- API pour interagir avec l'environnement d'exécution et pour fournir des routines utilitaires au programmeur d'application
- Utilisation d'une machine virtuelle interprétant byte code

Le langage Javacard

Possibilités du langage Java supportées par l'API JavaCard :

- Type supportés (byte, short, int), non supporté (char, long, double, float)
- Gestion des Exceptions
- Instanceof, super, this
- Mécanisme d'héritage, interface, modificateurs d'accès
- Méthode equals
- Allocation dynamique
- Méthodes virtuelles
- Tableau à une dimension
- Machine virtuelle : Byte code
- Objets stockés en EEPROM
- Objet Applet : toujours une applet active par défaut
- RMI (*depuis v2.2*) : permet de dispenses le programmeur de gestion des APDU

Le langage Javacard

L'API JavaCard ne permet pas d'exploiter l'ensemble des possibilités de Java :

- Types de données : long, double, float, char, string, tableaux multidimentions
- Chargement dynamique des classes
- Gestionnaire de sécurité
- Sérialisation des objets : Javacard gére une persistance par défaut dans EEPROM
- Clonage des objets
- Pas de threads
- Classe non supportées : Class, Byte, Boolean, Integer
- (avant Javacard 2.2) Pas de garbage collector ni de désallocation explicite
  - possibilité de perte de mémoire
  - Depuis V 2.2 : possibilité de l'activer volontairement
- *Format Bytecode modifié (optimisation), taille mots pile 16 bits, pas chargement dynamique*
- *APIs fotement limités : ss-ens de java.lang, algos crypto, nouveautés pour gérer apécifités carte, ...*

Squelette d'1 appli Javacard

1 Appli Java Card doit contenir classe étendant `javacard.framework.Applet`

```
public class MonApplet extends javacard.framework.Applet{

   public static void install(byte[] tab, short debut, byte long)
      throws ISOException {
      MonApplet monInstance = new MonApplet();
      ... }

   public boolean select(){
      // Initialisation des informations de session
      return true; }

   process(APDU apdu) throws ISOException{
      // On obtient le tableau contenant l'APDU de commande
      byte[] buffer = apdu.getBuffer();
      if ( buffer[ISO7816.OFFSET_CLA] == (byte)XX ) {
         switch (buffer[ISO7816.OFFSET_INS] ) {
            case YY:
               // Exécuter la commande et envoyer la réponse
               break;
               ...
}}}}
```


Squelette d'1 appli Javacard

Install

- invoquée une seule fois lors de l'installation de l'applet sur la carte.
- Contient allocations d'objets persistants de l'applet, dont applet
- plusieurs applets peuvent cohabiter sur une même carte Java et avant de leur envoyer une APDU de commande, l'entité dialoguant avec la carte doit d'abord envoyer une APDU de commande spécifique pour sélectionner l'applet à laquelle il souhaite envoyer les commandes suivantes.

Select

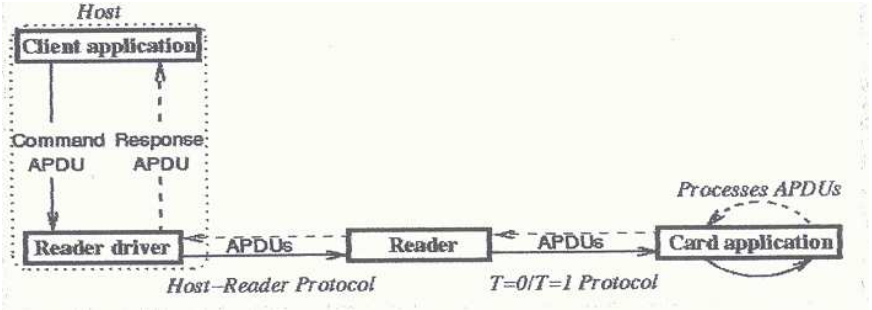
- La sélection se fait à l'aide d'un identifiant d'applet, ou AID attribué à l'installation
- Lorsqu'une applet est sélectionnée, l'environnement d'exécution invoque sa méthode select afin de lui permettre d'initialiser toutes les données de session.

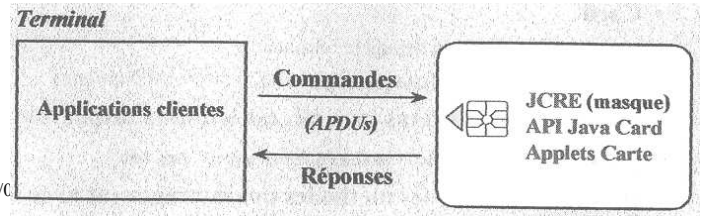
Process

- les APDU envoyées à la carte sont transmises à l'applet sélectionnées en invoquant sa méthode process
- L'applet peut alors analyser les octets de classe, d'instruction et éventuellement les paramètres P1 et P2 pour déterminer le traitement à effectuer avant d'envoyer une réponse

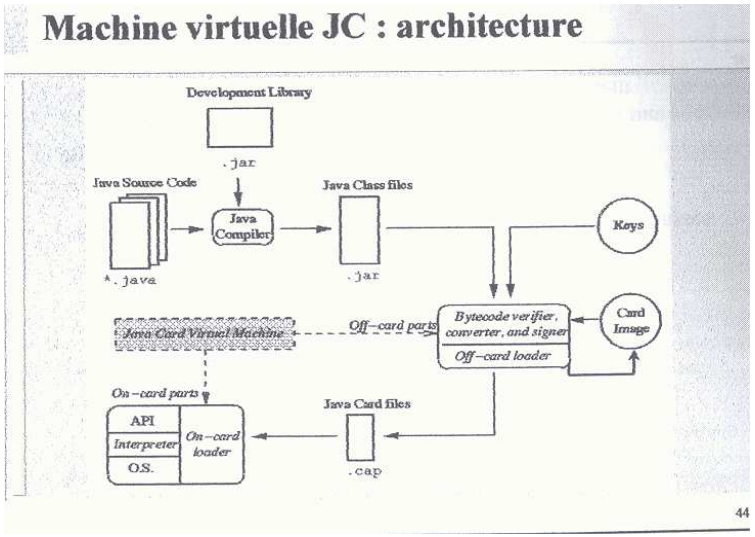
Archi d'une appli Javacard

Javacard





VM JC



Packetages & Extentions

Paquetages

- `Javacard.framework`
  - AID, APDU, Applet, ISO, PIN, System, Class
- `Java.lang`
- `Javacard.security`

Extensions

- `Javacardx.framework`
  - gestion de fichiers
- `Javacardx.crypto`
  - gestion clés privées, publiques, hashage, générateur aléatoire de clés
- `Javacardx.cryptoEnc`
  - algorithme de chiffrement DES

Agenda

Les Cartes à Puces

La JavaCard

Sécurité JavaCard

- **Signature / Vérif signature**
- **Génération clés symétriques**
- **Chiffrement / Déchiffrement symétrique**
- **Génération clés Asymétriques**
- **Chiffrement / Déchiffrement Asymétrique**

Signature messages

Code source java

```
public static byte[] creerSignature(byte[] message,PrivateKey pk)
{
   Signature s ;
   try
   {
      s = Signature.getInstance("MD5withRSA") ;
      s.initSign(pk) ;
      s.update(message) ;
      return s.sign() ;
   }
   catch (Exception e)
   {
      System.out.println("Exception :" + e) ;
   }
}
```

Vérification Signature

Code source java

```
public static boolean verifierSignature(byte[] message,byte[] signature,PublicKey pk)
{
   Signature s ;
   try
   {
      s = Signature.getInstance("MD5withRSA") ;
      s.initVerify(pk) ;
      s.update(message) ;
      return s.verify(signature) ;
   }
   catch (Exception e)
   {
      System.out.println("Exception :" + e) ;
   }
}
```


Génération clés symétriques

Code source java

```
public static Key generateDESKey()
{
   try
   {
      KeyGenerator keyGen = KeyGenerator.getInstance("DES") ;
      keyGen.init(56) ;
      return keyGen.generateKey() ;
   }
   catch (Exception e) System.out.println(e) ;
   return null ;
}
```


Chiffrement symétriques

Code source java

```
/**
* chiffrerMessageDES : fonction permettant de chiffrer un message
* en utilisant l'algorithme DES avec une clé de 56 bits
*/
public static byte[] chiffrerMessageDES(byte[] message, Key pk) throws Exception
{
   try
   {
      Cipher cipher = Cipher.getInstance("DES") ;
      cipher.init(Cipher.ENCRYPT_MODE, pk) ;
      return cipher.doFinal(message) ;
   }
   catch (Exception e) {System.out.println(e) ;}
   return null ;
}
```


Déchiffrement symétriques

Code source java

```
public static byte[] dechiffrerMessageDES(byte[] message, Key pk) throws Exception
{
   try
   {
      Cipher cipher = Cipher.getInstance("DES") ;
      cipher.init(Cipher.DECRYPT_MODE, pk) ;
      return cipher.doFinal(message) ;
   }
   catch (Exception e) {System.out.println(e) ;}
   return null ;
}
```

Génération clés Asymétriques

Code source java

```
public static KeyPair generateRSAKey()
{
   KeyPairGenerator kpg ;
   KeyPair kp ;
   try
   {
      kpg = KeyPairGenerator.getInstance("RSA") ;
   }
   catch (Exception e)
   {
      System.out.println("Exception : " + e) ;
   }
   kpg.initialize(1024) ;
   kp = kpg.generateKeyPair() ;
   if (kp==null) throw new Exception("Erreur création des 2 clés") ;
   return kp ;
}
```


Chiffrement Asymétriques

Code source java

```
public static byte[] chiffrerMessage(byte[] message,Key pk) {
   Provider p ;
   p=Security.getProvider("BC") ;
   try { if (p==null) java.security.Security.addProvider(new BouncyCastleProvider()) ; }
   catch (Exception e) {System.out.println(e) ;}
   Cipher cp ;
   try { cp = Cipher.getInstance("RSA","BC") ; }
   catch (Exception e) {System.out.println(e) ;}
   try {
      byte[] tmp1 ;   byte[] tmp2 =null ;
      cp.init(Cipher.ENCRYPT_MODE,pk) ;
      int blockSize = cp.getBlockSize() ;
      int debut = 0 ;
      for (int i = blockSize ; i < message.length ; i+= blockSize){
         cp.update(message,debut,blockSize) ;
         tmp1 = cp.doFinal() ;
         tmp2 = concat(tmp2,tmp1) ;
         debut += blockSize ;
      }
      return concat(tmp2,(cp.doFinal(message,debut,message.length - debut))) ;
   }
   catch (Exception e) {System.out.println(e) ;}
}
```


Déchiffrement Asymétriques

Code source java

```
public static byte[] dechiffrerMessage(byte[] message, Key pk) {
   Provider p ;
   p = Security.getProvider("BC") ;
   try { if (p == null) java.security.Security.addProvider (new BouncyCastleProvider()) ; }
   catch (Exception e) {System.out.println(e) ;}
   Cipher cp ;
   try { cp = Cipher.getInstance("RSA","BC") ; }
   catch (Exception e) {System.out.println(e) ;}
   try {
   byte[] tmp1 ;    byte[] tmp2 = null ;
   cp.init(Cipher.DECRYPT_MODE,pk) ;
   int blockSize = cp.getBlockSize() ;    int debut = 0 ;
   for (int i = blockSize ; i < message.length ; i+= blockSize) {
      cp.update(message,debut,blockSize) ;
      tmp1 = cp.doFinal() ;
      tmp2 = concat(tmp2,tmp1) ;
      debut += blockSize ;
   }
   return concat(tmp2,(cp.doFinal(message,debut,message.length - debut))) ;
   }
   catch (Exception e) {System.out.println(e) ;}
}
```