# Application Security – Lab 1
# Java Security and Cryptography

In this lab, you will have to complete several exercises using Java (J2SE 16 or before):

- Triple DES encryption
- RSA signature
- Using Class Loaders

You can answer in English or French to the questions, with complete sentences. You can work in teams of 2 students (not more).

Complete this document, zip it together with your Java code and send it to Yves.Roudier@univ-cotedazur.fr as follows:

- Zip it and rename it to **LabSession1-FirstNameStudent1-FirstNameStudent2.ZIP**
- Use the following subject: **[S&P] – Lab#1**
- Put in Cc: the second student if any.

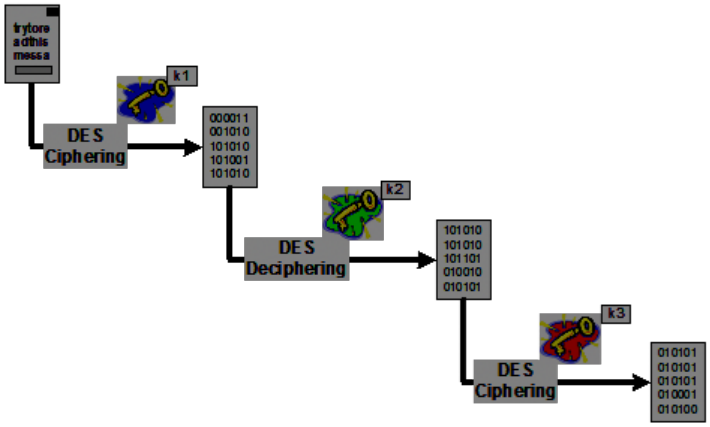
# Triple DES (10 points)

In this exercise, you have to answer to the questions in this document, and complete the file *./src/com/polytech/security/TripleDES.java*.

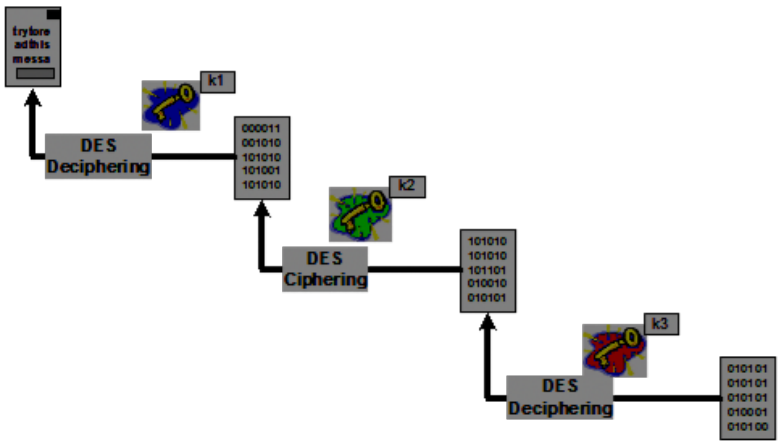
## *Background information*

3DES is a symmetric key block cipher, which applies three times the DES cipher/decipher algorithm.

3DES encryption is performed (in EDE mode) as depicted in the following figure:




3DES decryption is performed (in EDE mode) as depicted in the following figure:




Several keying configurations are possible with 3DES.

**In this lab session, K1, K2 and K3 will be independent.**

## *1. Data Encryption Standard (1 point)*

What is the size of DES cipher key's size? (0.5 point)

What are the size of the cipher blocks? (0.5 point)

## *2. DES/ECB/NoPadding (4,5 points)*

2.1. Explain ECB with diagram (0,5 point)

2.2. Explain NoPadding (0,5 point)

For the next two questions, you are asked to encrypt the following file:

*./ecb/clearTextFileECB*

2.3. Perform 3DES in ECB mode encryption in
`TripleDES ::private Vector encryptECB(...)` (1,75 points)

2.4. Perform 3DES in ECB mode decryption in
`TripleDES ::private void decryptECB(...)` (1,75 points)

## *3. DES/CBC/NoPadding (4,5 points)*

2.1. Explain CBC and its advantages over ECB with diagram (1 point)

For the two following questions, you are asked to encrypt the following file:

`./cbc/clearTextCBC`

2.2. Perform 3DES in CBC mode encryption in `TripleDES ::private Vector encryptCBC(...)`(2 points)

2.3. Perform 3DES in CBC mode decryption in `TripleDES ::private void decryptCBC(...)` (2 points)

# RSA Signature (10 points)

In this exercise, you have to answer to the questions into this document, and complete the file *./src/com/polytech/security/Asymetric.java*and *./src/com/polytech/security/Entity.java*

## *1. Use of Java Signature (3 points)*

### 1.1. Generation of a public/private key pair (1 point)

Complete method *Entity::Entity()*

- Generate a keypair generator object of type `java.security.KeyPairGenerator` for RSA.
- Generate a public/private keypair.
- Store them in class members `Entity::thePublicKey` and `Entity::thePrivateKey`.

### 1.2. RSA Signature (1 point)

Complete method *Entity::sign()*

- Create an signature object *java.security.signature* for « `SHA1withRSA` ».
- Initialize the object with the private key in *SIGN_MODE*.
- Sign

### 1.3. Check signature (1 point)

Complete method *Entity::checkSignature()*

- Create an objet *java.security.Signature*
- Initialize it in *VERIFY* mode with the public key
- Check the signature.

## *2. Implementation of your own RSA signature (5 points)*

### 2.1. Signature (2.5 points)

Complete method *Entity::mySign()* Implement your own signature using

- *javax.crypto.Cipher* with *RSA* in *ENCRYPT_MODE* mode
- *java.security.MessageDigest* with *SHA1*.

### 2.2. Check signature (2.5 points)

Complete method *Entity::myCheckSignature()*

Implement your own signature verification using:

- `javax.crypto.Cipher` with `RSA` in `DECRYPT_MODE` mode
- `java.security.MessageDigest` with `SHA1`

## *3. RSA Ciphering (2 points)*

**Warning : Oracle's RSA implementation does not support messages greater than 127 bytes.**

### 3.1. RSAEncryption (1 point)

Complete method `Entity::encrypt()`

### 3.2. RSADecryption (1 point)

Complete method `Entity::decrypt()`.

## *3. Secure session key exchange (2 points)*

You have to implement the following protocol between Alice and Bob for a secure session key exchange.

1. Alice sends her public key to Bob.
2. Bob generates a DES session key.
3. Bob encrypts it with Alice's public key.
4. Alice decrypts the DES key with her private key.
5. Alice sends a message to Bob with her session key
6. Bob decrypts the message with the session key.

Fill the static method ***KeyExchangeProtocol()*** in ***Asymetric.java***.

You can also refer to the slides from the application security lecture for further details on this secure session key exchange.

# Class Loaders
# (10 points)

## 1. Using the primordial class loader (2 points)

Using the forName() method of class java.lang.Class, you will dynamically load a class in the program.

First create four classes Car, Truck, Bicycle and Motorcycle each with a ride() method. They can for instance implement the same interface Vehicle.

Now load the first vehicle class in your program, and call its method.

## 2. Using the RMI class loader for local files (2 points)

Now, use class RMIClassLoader and its method loadClass(...) to load the second local vehicle class.

## 3. Using the RMI class loader for remote files (2 points)

You will now use class RMIClassLoader for loading the third vehicle class that you will have put on a local webserver for use through RMI.

Hint: you will also need to set the security manager (think about RMISecurityManager).

## 4. Writing a custom class loader (4 points)

We propose you to analyze how a custom class loader is written. Class MyClassLoader loads files from a subdirectory called "Motorbikes". Move the Motorcycle.class file to this directory and try to load it using this custom class loader.

Next, have a look at the contents of the class loader.