Institut Galilée - Université Paris 13

Rapport

Projet - sécurité et théorie de l'information

Étudiante :
Tiziri OULD HADDA

Enseignants :
M. Olivier BODINI,
M. Sergey DOVGAL

27 mai 2020

Table des matières

Architecture du code

Le code du projet est réalisé avec la version Java 8 et ne contient pas de librairies extérieurs à JAVA SE. Le package "crypto" contient :

— **un fichier source Main.java** qui contient une fonction pour chaque réponse aux questions de l'énoncé.

```
public static void main(String[] args) {
    //premiere partie
    question1_1();end_question();
    question1_2();end_question();
    question1_3();end_question();
    question1_4();end_question();
    question1_5();end_question();
    question1_6();end_question();
    question1_7();end_question();
    question1_8();end_question();
    //deuxieme partie
    question2_1();end_question();
    question2_2();end_question();
    question2_3();end_question();
    question2_4();end_question();
  }
```

Remarque : Toutes les constantes sont nommées (en majuscule) en fonction de la question pour plus de clarté dans le code :

```
  private static final String Q1_1_HEXTOBASE64      = "49276d20...736872 6f6f6d";

  private static final String Q1_2_FIXEDXOR_HEXSTR1 = "1c011100...009181c";
  private static final String Q1_2_FIXEDXOR_HEXSTR2 = "6869742...6c277320657965";

  private static final String Q1_3_BYTEXOR_HEXSTR1  = "1b373733...783a393b3736";

  private static final String Q1_4_FILENAME     = "detecting-singlechar-xor.txt";
  ...
```

— **un package cipher** : Contient les code source des algorithme de cryptographie utilisés.

— **un package tools** : qui contient une Classe permettant de faire les conversions Hexadécimal vers et depuis des Strings, d'un tableau d'octets (byte[]) et Base64. Et aussi une autre qui permet de calculer les metriques demandé dans le projet.

— **un package examples** : utilisée afin de chargé plus facilement les fichier exemples.

Compilation et Exécution :

— le projet à été créé avec la version Java 8.

— Il faut faire attention au package lors de l'execution.

— Pour eviter toutes confusion, on peut faire la compilation et l'éxecution en utilisant le Makefile présent dans le projet.

  — Pour compiler : *make*

  — Pour éxecuter : *make run*

  — Pour purger les objets : *make clean*

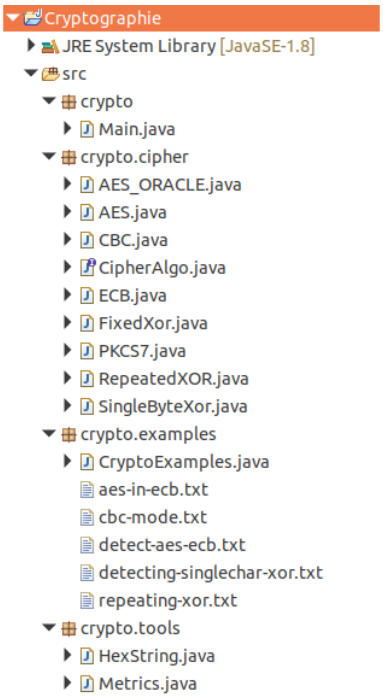

FIGURE 1 – Package Java et Structure

Chaque algorithme de chiffrement utilisé est représenté par une Class dédié qui herite de la Class CipherAlgo suivante :

```
package crypto.cipher;

public interface CipherAlgo {
  public byte[] encrypt(byte[] plaintext_bytes);
  public byte[] decrypt(byte[] ciphertext_bytes);
}
```

Celle ci sera utilisé par la suite (ECB, CBC) afin de généralisé (utilisé comme type générique) la notion de chiffrement.

Concernant le chargement des exemples, la classe suivante cherche dans le même package(voir package examples - Figure 1) ces fichiers (ressources) et renvoi par moyen de méthodes statique (ne nécessite pas d'instance) le stream ou le contenu associé .

```
package crypto.examples;
import ...

public class CryptoExamples {
   public static InputStream getInputStream(String filename) {
      InputStream inputStream = null;
          try {
            inputStream = CryptoExamples.class.getResourceAsStream(filename);
          } catch (Exception e) {
              System.out.println(e.getMessage());
          }
          return inputStream;
      }
   ...
}
```

1 Qualifying set

1.1 Convert hex to base64

Pour faire cette conversion à partir d'une chaine de caractère, on doit faire d'abort sa conversion vers un tableau d'octects. Ce tableau sera le résultat d'un parsing des caractère deux à deux, car **un octet représente deux digit dans la base hexadécimal**.

Une fois transformé en octects on **utilisera la classe Base64 de Java pour les coder.**

```
public class HexString {
    ...
  public static String toBase64( String hexStr) {
    byte[] hexBytes = new byte[hexStr.length() / 2];
    for (int i = 0; i < hexBytes.length; i++) {
         int index = i * 2;
         int integer = Integer.parseInt(hexStr.substring(index, index + 2), 16);
         hexBytes[i] = (byte) integer;
    }
     return Base64.getEncoder().encodeToString(hexBytes);
  }
```

On appliquant cette fonction à la String donnée dans la première question on trouve.

Resultat:

```
Question 1.1 : Convert hex to base64
Original: 49276
    d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d

Result   : SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t
```

1.2 Fixed XOR

Le Fixed XOR fait tout simplement un **XOR octet par octect de deux tableaux** (<message clair> XOR <clé>).

Remarques :

— La fonction de décodage pointe sur cette même fonction.

— Il est a noter aussi que cette fonction **renvoi null si les deux tableau n'ont pas la même taille**.

```
public class FixedXor {
  public static byte[] encode(byte[] bytes, byte[] key) {
    if(bytes.length != key.length) {
      return null;
    }
    byte[] result_bytes = new byte[bytes.length];
    for(int i = 0; i < bytes.length; i++) {
      result_bytes[i] = (byte)(bytes[i] ^ key[i]);
    }
    return result_bytes;
  }
  ...
```

Resultat:

```
Question 1.2 : Fixed XOR
Original: 1c0111001f010100061a024b53535009181c
Original: 686974207468652062756c6c277320657965
Result  : 746865206b696420646f6e277420706c6179
```

1.3 Single-byte XOR cipher

Chiffrement et déchiffrement :

Pour implémenter cela, on réalise un XOR du caractère clé avec tout les octets du tableau bytes : qui représente ici notre message.

```
public class SingleByteXor {
  public static byte[] encode(byte[] bytes, byte key) {
    byte[] result_bytes = new byte[bytes.length];

    for(int i = 0; i < bytes.length; i++) {
      result_bytes[i] = (byte)(bytes[i] ^ key);
    }
    return result_bytes;
  }
  ...
```

Detection de la clé :

Afin de detecter le caractère ayant encodé notre message nous aurons besoin de décoder le message avec tout les 256 caractère possible, puis de prendre celui qui donne un chiffrement maximisant la metrique suivante.

Cette metrique utilise un tableau qui représent la fréquence d'apparition moyenne de chaque caractère dans la langue Anglaise, voir colonne "English" dans le tableau du lien : `https://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_the_English_language`

Dans le cas de l'anglais on représentera que l'alphabet et le caractère espace. Les autres caractère ont tous un score nul. Concernant le caractère espace, sa fréquence doit être supérieure (pas de beaucoup) à de la lettre "e" (fréquence max dans l'alphabet). *Cette metrique n'est pas sensible à la case* donc on met le message on miniscule pour eviter toutes confusion.

```
public class Metrics {
  private static final double[] ALPHA_FREQ_ENGLISH ={
      8.167, //a
      1.492, //b
            ...
      0.074  //z
      //others = 0
    };
  private static final double   SPACE_FREQ_ENGLISH = 20;
  public static double english_frequency_score(String str) {
    double valeur = 0;
    for(byte b : str.toLowerCase().getBytes()) {
      if(b >= 'a' && b <= 'z')
        valeur+=ALPHA_FREQ_ENGLISH[b-'a'];
      else if(b == ' ') {
        valeur+=SPACE_FREQ_ENGLISH;
      }
    }
    return valeur;
  }
```

On appliquant l'algorithme suivant :

```
public static byte find_key(byte[] bytes) {
  byte[] decoded_bytes;
  double score_max = -1;
  byte key = 0; //null char
  for(int i = 0; i < 256; i++)
  {
    decoded_bytes = decode(bytes, (byte)i);
    String englishDecoded = new String(decoded_bytes);
    double score = Metrics.english_frequency_score(englishDecoded);
    if(score > score_max) {
      score_max = score;
      key = (byte)i;
    }
  }
  return key;
}
```

Resultat:

```
Question 1.3 : Single-byte XOR cipher
Found key using english word frequency metric is : X
Decoded message using this key : Cooking MC's like a pound of bacon
```

1.4 Detect single-character XOR

L'objectif de cette question et de pouvoir détecter parmi un ensemble de messages chiffrés, celui qui semble être chiffré avec "Single-character XOR". Pour le trouver on cherche la clé de chaque message puis en les décryptant on choisi celui dont le score (english frequency) est maximal.

```
public static void question1_4() {
  System.out.println("Question 1.4 : Detect single-character XOR");
  String line = null;
  String plaintext = "";
  String ciphertext = "";
  int count = 1, line_num = 1;
  double score_max = -1;
  try {
    System.out.println("Read file : " + Q1_4_FILENAME);
    InputStream in =  CryptoExamples.getInputStream(Q1_4_FILENAME);
    BufferedReader input = new BufferedReader(new InputStreamReader(in,"UTF-8"));
    byte[] decoded_bytes;
    String english_text;
    while ((line = input.readLine()) != null) {
      byte key = SingleByteXor.find_key(HexString.toBytes(line));
      decoded_bytes = SingleByteXor.decode(HexString.toBytes(line), (byte)key);
      english_text  = new String(decoded_bytes);
      double score = Metrics.english_frequency_score(english_text);
      if(score > score_max) {
        line_num   = count;
        score_max  = score;
        ciphertext = line;
        plaintext  = english_text;
      }
      count++;
    }
    System.out.println("line "+ line_num + " (has maximum score), ciphertext:" +
  ciphertext);
    System.out.println("line "+ line_num + " (decoded message  ), plaintext :" +
  plaintext);
    input.close();
  }catch (Exception e) {
    System.out.println(e.getMessage());
  }
}
```

Remarque : A noter que ***Q1_4_FILENAME*** est nom de fichier "detecting-singlechar-xor.txt" ;

Resultat:

```
Question 1.4 : Detect single-character XOR
Read file : detecting-singlechar-xor.txt
line 171 (has maximum score), ciphertext:7
    b5a4215415d544115415d5015455447414c155c46155f4058455c5b523f
line 171 (decoded message  ), plaintext :Now that the party is jumping
```

1.5 Implement repeating-key XOR

Pour le repeating-key XOR, on utilise l'opération MODULO pour repeter la clé.

```
public class RepeatedXOR {
  public static byte[] encode(byte[] bytes, byte[] key) {
    byte[] result_bytes = new byte[bytes.length];
    for(int i = 0; i < bytes.length; i++) {
      result_bytes[i] = (byte) (bytes[i] ^ key[i % key.length]);
    }
    return result_bytes;
  }
```

Resultat:

```
Question 1.5 : Implement repeating-key XOR
plaint text  : Burning 'em, if you ain't quick and nimble I go crazy when I hear a cymbal
key          : ICE
result       : 0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a2622632427276527
2a282b2f20690a652e2c652a3124333a653e2b2027630c692b20283165286326302e27282f
```

1.6 Break repeating-key XOR

1.6.1 Hamming distance test

Dans la première partie de la question, il est demandé d'implémenter la distance de hamming. Pour implémenter cette metrique il faudra faire un **XOR octet par octet** des deux messages donnés **pour detecter les bits differents**. Puis compter sur chaque octect résultant ces bits en utilisant l'operation de décalage et de ET binaire.

```
  public static int hamming_distance(byte[] bytes1, byte[] bytes2) {
    if(bytes1.length != bytes2.length) {
      return -1;
    }
    int distance = 0;
    byte[] tmp = FixedXor.encode(bytes1, bytes2);
    for(int i = 0; i < tmp.length; i++) {
      for(int j = 0; j < 8; j++) {
        if( ( (tmp[i] >> j) & 1) == 1) {
          distance++;
        }
      }
    }
    return distance;
  }
```

On appliquant cette méthode sur les deux chaine de caractère donné dans la question 1.6, on trouve :

Resultat:

```
Question 1.6 : Break repeating-key XOR
Text 1       : this is a test
Text 2       : wokka wokka!!!
Test hamming distance: 37
```

1.6.2 Breaking algorithm

C'est l'algorithme décrit dans l'ennocé. Le code étant un peu long, j'ai décider d'expliquer ses différentes étapes :

- Etape 0 : Le fichier en entrée étant en Base64 il faudra donc le décoder. Le fonction decode de la classe Base64 donne en retour un tableau (une suite) d'octets.
- Etape 1 : Trouver la taille de la clé.
  - on itère sur un interval de taille possible (KEYSIZE : 2->40), puis on calcule la distance de hamming moyenne à partir de quatres premières séquence de taille KEYSIZE.
  - le KEYSIZE recherché correspond à celle qui minimize la distance de hamming.
- Etape 3 : **découper le contenu du fichier par blocs de KEYSIZE**, cela veux dire que **chaque bloc est le résultat d'un chiffrement par un Fixed XOR** (avec la clé qu'on cherche).
- Etape 4 : **l'operation de transpose** va permettre de mettre tous les caractère chiffré avec le même caractère de la clé ensemble => **maintenant chaque bloc est le résultat d'un chiffrement par Single-character XOR (k=cle[i], i numéro du block)**.
- Etape 5 : chercher la clé : on sait trouvé la clé d'un Single-character XOR, on procède par iteration sur les derniers bloc pour trouver toute la clé.

Resultat:

```
----- Break repeating-key XOR -----
----- Read file : repeating-xor.txt -----
----- Getting minimum hamming distance from average of 4 sequences-----
----- Optimal value found for KEYSIZE: 29 -----
The key found is: Terminator X: Bring the noise
decrypted plaintext: I'm back and I'm ringin' the bell
A rockin' on the mike while the fly girls yell
In ecstasy in the back of me
Well that's my DJ Deshay cuttin' all them Z's
Hittin' hard and the girlies goin' crazy
Vanilla's on the mike, man I'm not lazy.

I'm lettin' my drug kick in
It controls my mouth and I begin
To just let it flow, let my concepts go
My posse's to the side yellin', Go Vanilla Go!

Smooth 'cause that's the way I will be
And if you don't give a damn, then
Why you starin' at me
So get off 'cause I control the stage
There's no dissin' allowed
I'm in my own phase
The girlies sa y they love me and that is ok
And I can dance better than any kid n' play

Stage 2 -- Yea the one ya' wanna listen to
It's off my head so let the beat play through
So I can funk it up and make it sound good
1-2-3 Yo -- Knock on some wood
```

```
For good luck, I like my rhymes atrocious
Supercalafragilisticexpialidocious
I'm an effect and that you can bet
I can take a fly girl and make her wet.

I'm like Samson -- Samson to Delilah
There's no denyin', You can try to hang
But you'll keep tryin' to get my style
Over and over, practice makes perfect
But not if you're a loafer.

You'll get nowhere, no place, no time, no girls
Soon -- Oh my God, homebody, you probably eat
Spaghetti with a spoon! Come on and say it!

VIP. Vanilla Ice yep, yep, I'm comin' hard like a rhino
Intoxicating so you stagger like a wino
So punks stop trying and girl stop cryin'
Vanilla Ice is sellin' and you people are buyin'
'Cause why the freaks are jockin' like Crazy Glue
Movin' and groovin' trying to sing along
All through the ghetto groovin' this here song
Now you're amazed by the VIP posse.

Steppin' so hard like a German Nazi
Startled by the bases hittin' ground
There's no trippin' on mine, I'm just gettin' down
Sparkamatic, I'm hangin' tight like a fanatic
You trapped me once and I thought that
You might have it
So step down and lend me your ear
'89 in my time! You, '90 is my year.

You're weakenin' fast, YO! and I can tell it
Your body's gettin' hot, so, so I can smell it
So don't be mad and don't be sad
'Cause the lyrics belong to ICE, You can call me Dad
You're pitchin' a fit, so step back and endure
Let the witch doctor, Ice, do the dance to cure
So come up close and don't be square
You wanna battle me -- Anytime, anywhere

You thought that I was weak, Boy, you're dead wrong
So come on, everybody and sing this song

Say -- Play that funky music Say, go white boy, go white boy go
play that funky music Go white boy, go white boy, go
Lay down and boogie and play that funky music till you die.

Play that funky music Come on, Come on, let me hear
Play that funky music white boy you say it, say it
Play that funky music A little louder now
Play that funky music, white boy Come on, Come on, Come on
Play that funky music
```

1.7 AES in ECB mode

Création de la classe AES :

L'algorithme de chiffrement AES implémente l'interface **CipherAlgo**. La classe AES est crée en utilisant la **Class Cipher de Java**, et comme paramètre, j'ai choisi de l'instancier avec le mode "AES/ECB/NoPadding". Le faite de ne pas choisir le mode "AES" est tout simplement car il lève une exception dans la méthode de déchiffrement(voir image - ci-dessous).

Cela est du au faite que le seul moyen de spécifié le padding (ou non padding) est de choisir le mode "AES/ECB/NoPadding" et d'après la doc Java c'est le mode par défaut de "AES" (il fait exactement ce que fait AES).

```
Question 2.2 : Implement CBC mod
----- Create AES Instance using key : YELLOW SUBMARINE -----
----- Create CBC associated to ASE Instance -----
----- Decrypt file : cbc-mode.txt -----
javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
        at com.sun.crypto.provider.CipherCore.unpad(CipherCore.java:975)
        at com.sun.crypto.provider.CipherCore.fillOutputBuffer(CipherCore.java:1056)
        at com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:853)
        at com.sun.crypto.provider.AESCipher.engineDoFinal(AESCipher.java:446)
```

FIGURE 2 – Exception levé par la méthode decrypt AES (doFinal) : dans de nombreux forum Java, ils disent qu'il faut mentionné explécitement s'il y a un Padding et c'est le seul moyen trouvé dans la doc Java

Voir aussi : *https://stackoverflow.com/questions/7615743/java-aes-without-padding*

Code de la classe AES :

On remarquera ici que la **clé** est initialisé dans le constructeur de la classe AES et on ne prend que les 16 premier octects. C'est important car **c'est la taille de la clé qui défini la taille du bloc**.

```
public class AES implements CipherAlgo {
    protected static final String CIPHER_MODE  = "AES/ECB/NoPadding";
    protected static final int    KEY_SIZE     = 16;
    protected static final int    BLOCKSIZE    = 16;
    protected byte[] key_bytes;
  public AES(String key_str) {
    this.key_bytes = Arrays.copyOf(key_str.getBytes(), KEY_SIZE);;
  }
    ...
    public byte[] decrypt(byte[] content) {
        try {
          SecretKeySpec key = new SecretKeySpec(key_bytes, "AES");
            Cipher cipher = Cipher.getInstance(CIPHER_MODE);
            cipher.init(Cipher.DECRYPT_MODE, key);
            byte[] result = cipher.doFinal(content);
            return result;
        }
        ...
```

Implémentation de l'algorithme ECB :

L'implémentation de ECB est très simple, cette dernière necessite de comprendre seulement son shéma. Afin de rendre l'algorithme générique j'ai rajouté un attribut **algo** qui permet de specifié l'algorithme de chiffrement utilisé. Ainsi vu que nos algorithme implémente l'interface CiperAlgo, on l'utilisera ici comme type.

Ci dessous le code de la classe ECB avec celui de la méthode encrypt. ECB implémente CipherAlgo :

```java
public class ECB implements CipherAlgo {
  private CipherAlgo algo;
  //private int block_size;
  public static final int BLOCKSIZE = 16;
  public ECB(CipherAlgo algo, int block_size) {
    this.algo = algo;
    //this.block_size = block_size;
  }
    ...
  public byte[] encrypt(byte[] plaintext_bytes) {
    int nb_blocks = plaintext_bytes.length / BLOCKSIZE;
    byte[] plaintext_block, output_bytes;
    byte[] result = new byte[nb_blocks * BLOCKSIZE];
    for(int i = 0; i < nb_blocks; i++) {
      plaintext_block   =  Arrays.copyOfRange(plaintext_bytes, i * BLOCKSIZE, (i+1)*
    BLOCKSIZE);
      output_bytes    =  algo.encrypt(plaintext_block);
      add_block_result(result, output_bytes, i);
    }
    return result;
  }
```

Remarque :

- Pour la fonction "decrypt", il suffit juste de remplacer l'appel a la méthode "algo.encrypt" par "algo.decrypt" car c'est le même shéma.
- l'inconvénient dans cette méthode c'est que si jamais le dernier block n'est pas complétement rempli, il n'est donc pas pris par l'algorithme (car *nb_block* est le résultat d'une division entière).
- **Important :** Vu qu'on connait à l'avance la taille du résultat, on utilise la méthode **add_block_result** pour positionné le résultat **"output_bytes"** du chiffrement/déchiffrement à la position **"i"** avant de retourné **"result"**.
- L'appel a cette classe se fait comme suit (de même pour CBC plus tard) :

```java
    System.out.println("----- Create AES Instance using key : " + Q1_7_KEY_STR + "
     -----");
  AES aes = new AES(Q1_7_KEY_STR);
  System.out.println("----- Create ECB associated to ASE Instance -----");
  ECB aes_128_ebc = new ECB(aes, 16);
  System.out.println("----- Decrypt file : " + Q1_7_FILENAME + " -----");
  byte[] plaintext_bytes = aes_128_ebc.decrypt(ciphertext_bytes);
  System.out.println(new String(plaintext_bytes));
```

Resultat:

```
Question 1.7 : AES in ECB mode
----- Create AES Instance using key : YELLOW SUBMARINE -----
----- Create ECB associated to ASE Instance -----
----- Decrypt file : aes-in-ecb.txt -----
I'm back and I'm ringin' the bell
A rockin' on the mike while the fly girls yell
In ecstasy in the back of me
Well that's my DJ Deshay cuttin' all them Z's
Hittin' hard and the girlies goin' crazy
Vanilla's on the mike, man I'm not lazy.

I'm lettin' my drug kick in
It controls my mouth and I begin
To just let it flow, let my concepts go
My posse's to the side yellin', Go Vanilla Go!

Smooth 'cause that's the way I will be
And if you don't give a damn, then
Why you starin' at me
So get off 'cause I control the stage
There's no dissin' allowed
```

```
I'm in my own phase
The girlies sa y they love me and that is ok
And I can dance better than any kid n' play

Stage 2 -- Yea the one ya' wanna listen to
It's off my head so let the beat play through
So I can funk it up and make it sound good
1-2-3 Yo -- Knock on some wood
For good luck, I like my rhymes atrocious
Supercalafragilisticexpialidocious
I'm an effect and that you can bet
I can take a fly girl and make her wet.

I'm like Samson -- Samson to Delilah
There's no denyin', You can try to hang
But you'll keep tryin' to get my style
Over and over, practice makes perfect
But not if you're a loafer.

You'll get nowhere, no place, no time, no girls
Soon -- Oh my God, homebody, you probably eat
Spaghetti with a spoon! Come on and say it!

VIP. Vanilla Ice yep, yep, I'm comin' hard like a rhino
Intoxicating so you stagger like a wino
So punks stop trying and girl stop cryin'
Vanilla Ice is sellin' and you people are buyin'
'Cause why the freaks are jockin' like Crazy Glue
Movin' and groovin' trying to sing along
All through the ghetto groovin' this here song
Now you're amazed by the VIP posse.

Steppin' so hard like a German Nazi
Startled by the bases hittin' ground
There's no trippin' on mine, I'm just gettin' down
Sparkamatic, I'm hangin' tight like a fanatic
You trapped me once and I thought that
You might have it
So step down and lend me your ear
'89 in my time! You, '90 is my year.

You're weakenin' fast, YO! and I can tell it
Your body's gettin' hot, so, so I can smell it
So don't be mad and don't be sad
'Cause the lyrics belong to ICE, You can call me Dad
You're pitchin' a fit, so step back and endure
Let the witch doctor, Ice, do the dance to cure
So come up close and don't be square
You wanna battle me -- Anytime, anywhere

You thought that I was weak, Boy, you're dead wrong
So come on, everybody and sing this song

Say -- Play that funky music Say, go white boy, go white boy go
play that funky music Go white boy, go white boy, go
Lay down and boogie and play that funky music till you die.

Play that funky music Come on, Come on, let me hear
Play that funky music white boy you say it, say it
Play that funky music A little louder now
Play that funky music, white boy Come on, Come on, Come on
Play that funky music
```

1.8 Detect AES in ECB mode

Pour détécter le mode ECB, il faudra decoupé le message chiffré en plusieurs block de taille 16 octet (128bits), puis comparée de block a block (i != j) s'il y a une répétition.

```
public class ECB implements CipherAlgo {
    ...
  public static boolean detect_ecb(byte[]cipherbytes) {
    int    nb_blocks    = cipherbytes.length/BLOCKSIZE;
    byte[][] cipher_blocks  = new byte[nb_blocks][BLOCKSIZE];
    int k = 0;
    for(int i = 0; i < nb_blocks; i++){
      for(int j = 0; j < BLOCKSIZE; j++){
        cipher_blocks[i][j] = cipherbytes[k];
        k++;
      }
    }
    for(int i = 0; i < nb_blocks; i++) {
      for(int j = 0; j < nb_blocks; j++) {
        if(Arrays.equals(cipher_blocks[i], cipher_blocks[j]) && i != j){
          return true;
        }
      }
    }
    return false;
  }
```

Le résultat trouvé pour fichier "detect-aes-ecb.txt" est le suivant :

Resultat:

```
Question 1.8 : Detect AES in ECB mode
----- Read file : detect-aes-ecb.txt -----
ASE ECB line number: 133, text:
d880619740a8a19b7840a8a31c810a3d08649af70dc06f4fd5d2d69c744cd283e2dd052f6b641dbf9d11b034
8542bb5708649af70dc06f4fd5d2d69c744cd2839475c9dfdbc1d46597949d9c7e82bf5a08649af70dc06f4f
d5d2d69c744cd28397a93eab8d6aecd566489154789a6b0308649af70dc06f4fd5d2d69c744cd283d403180c
98c8f6db1f2a3f9c4040deb0ab51b29933f2c123c58386b06fba186a
```

2 Block crypto

2.1 Implement PKCS7 padding

Le PKCS7 paddding consiste à rajouter des octets pour remplir le dernier bloc incomplet. Le octets rajouter sont au faite une répétition du nombre d'octet manquant.

Exemple :

- "YELLOW SUBMARINE" contient 16 octets.
- Pour compléter jusqu'à 20 octets => donc il y a 4 octets manquants.
- le padding a rajouter (en hexadécimal) est donc le suivant : 04040404

```
public class PKCS7 {
  ...
  public static byte[] padding(byte[] input, int block_size) {
    if(input.length % block_size == 0) {
      return input;
    }
    int nb_blocks = (input.length / block_size) + 1;
    byte[] result = new byte[nb_blocks * block_size];
    byte pad = (byte) (result.length - input.length);
    for(int i=0; i< result.length; i++) {
      if(i < input.length)
        result[i] = input[i];
      else
        result[i] = pad;
    }
    return result;
  }
}
```

Resultat:

```
Question 2.1 : Implement PKCS#7 padding
Original          : YELLOW SUBMARINE
Original hex      : 59454c4c4f57205355424d4152494e45
PKCS7 padding hex: 59454c4c4f57205355424d4152494e4504040404
```

2.2 Implement CBC mode

Même choix d'implémentation avec le mode ECB. A noter aussi **Remarques sur la fonction de chiffrement :**

- le vecteur initiale peut se configurer avec la méthode *set_init_vector*.
- dans la première itération on chiffre le bloc actuel avec le vecteur initial.
- en itérant sur les blocs du message initial, on chiffre le résultat du xor entre le bloc actuel et le block chiffré précédement.

```
public class CBC implements CipherAlgo {
  private CipherAlgo algo;
  private byte[]  init_vector;
  public static final int BLOCKSIZE = 16;
  ...
  public CBC(CipherAlgo algo, int BLOCKSIZE) {
        ...
    }
    ...
  public byte[] encrypt(byte[] plaintext_bytes) {
    int nb_blocks = plaintext_bytes.length / BLOCKSIZE;
    byte[] plainttext_block, output_bytes = init_vector;
    byte[] result = new byte[nb_blocks * BLOCKSIZE];
    for(int i = 0; i < nb_blocks; i++) {
      plainttext_block =  Arrays.copyOfRange(plaintext_bytes, i * BLOCKSIZE, (i+1) *
    BLOCKSIZE);
      plainttext_block =  FixedXor.encode(plainttext_block, output_bytes);
      output_bytes     =  algo.encrypt(plainttext_block);
      add_block_result(result, output_bytes, i);
    }
    return result;
  }
```

la fonction de déchiffrement : Pour cette fonction, j'ai fait le chemin inverse du shéma CBC : c'est a dire il faut faire le XOR à la fin.

```
  public byte[] decrypt(byte[] ciphertext_bytes) {
    int nb_blocks = ciphertext_bytes.length / BLOCKSIZE;
    byte[] ciphertext_block_current, ciphertext_block_prev, xor_input, plaintext_bytes;
    byte[] result = new byte[nb_blocks * BLOCKSIZE];
    for(int i = nb_blocks - 1; i >= 0; i--) {
      ciphertext_block_current =  Arrays.copyOfRange(ciphertext_bytes, i * BLOCKSIZE, (i
    +1) * BLOCKSIZE);
      if(i == 0) {
        ciphertext_block_prev    =  init_vector;
      }else {
        ciphertext_block_prev    =  Arrays.copyOfRange(ciphertext_bytes, (i-1) *
    BLOCKSIZE, (i)*BLOCKSIZE);
      }
      xor_input     =  algo.decrypt(ciphertext_block_current);
      plaintext_bytes =  FixedXor.encode(xor_input, ciphertext_block_prev);
      add_block_result(result, plaintext_bytes, i);
    }
    return result;
  }
```

Application à la question 2.2 :

Resultat:

```
Question 2.2 : Implement CBC mod
----- Create AES Instance using key : YELLOW SUBMARINE -----
----- Create CBC associated to ASE Instance -----
----- Decrypt file : cbc-mode.txt -----
I'm back and I'm ringin' the bell
A rockin' on the mike while the fly girls yell
In ecstasy in the back of me
Well that's my DJ Deshay cuttin' all them Z's
Hittin' hard and the girlies goin' crazy
Vanilla's on the mike, man I'm not lazy.

I'm lettin' my drug kick in
It controls my mouth and I begin
To just let it flow, let my concepts go
My posse's to the side yellin', Go Vanilla Go!

Smooth 'cause that's the way I will be
And if you don't give a damn, then
Why you starin' at me
So get off 'cause I control the stage
There's no dissin' allowed
I'm in my own phase
The girlies sa y they love me and that is ok
And I can dance better than any kid n' play

Stage 2 -- Yea the one ya' wanna listen to
It's off my head so let the beat play through
So I can funk it up and make it sound good
1-2-3 Yo -- Knock on some wood
For good luck, I like my rhymes atrocious
Supercalafragilisticexpialidocious
I'm an effect and that you can bet
I can take a fly girl and make her wet.

I'm like Samson -- Samson to Delilah
There's no denyin', You can try to hang
But you'll keep tryin' to get my style
Over and over, practice makes perfect
But not if you're a loafer.

You'll get nowhere, no place, no time, no girls
Soon -- Oh my God, homebody, you probably eat
Spaghetti with a spoon! Come on and say it!

VIP. Vanilla Ice yep, yep, I'm comin' hard like a rhino
Intoxicating so you stagger like a wino
So punks stop trying and girl stop cryin'
Vanilla Ice is sellin' and you people are buyin'
'Cause why the freaks are jockin' like Crazy Glue
Movin' and groovin' trying to sing along
All through the ghetto groovin' this here song
Now you're amazed by the VIP posse.

Steppin' so hard like a German Nazi
Startled by the bases hittin' ground
There's no trippin' on mine, I'm just gettin' down
Sparkamatic, I'm hangin' tight like a fanatic
You trapped me once and I thought that
You might have it
So step down and lend me your ear
'89 in my time! You, '90 is my year.

You're weakenin' fast, YO! and I can tell it
Your body's gettin' hot, so, so I can smell it
So don't be mad and don't be sad
'Cause the lyrics belong to ICE, You can call me Dad
You're pitchin' a fit, so step back and endure
Let the witch doctor, Ice, do the dance to cure
So come up close and don't be square
You wanna battle me -- Anytime, anywhere
```

```
You thought that I was weak, Boy, you're dead wrong
So come on, everybody and sing this song

Say -- Play that funky music Say, go white boy, go white boy go
play that funky music Go white boy, go white boy, go
Lay down and boogie and play that funky music till you die.

Play that funky music Come on, Come on, let me hear
Play that funky music white boy you say it, say it
Play that funky music A little louder now
Play that funky music, white boy Come on, Come on, Come on
Play that funky music
```

2.3 An ECB/CBC detection oracle

ECB/CBC oracle encryption est désigné par la classe AES_ORACLE. Concernant son implémentation j'ai décidé qu'elle soit hérité de la classe AES ce qui fait d'elle aussi une classe implémentant CipherAlgo.

```
public class AES_ORACLE extends AES {
  public AES_ORACLE() {
    super();
    new Random().nextBytes(key_bytes);
  }
    ...
  public byte[] oracle_encrypt(byte[] input_bytes) {
    System.out.println("Random generated Key in hex : " + HexString.fromBytes(key_bytes))
    ;
    byte[] init_vect = new byte[BLOCKSIZE];
    int a = new Random().nextInt(6) + 5;
    int b = new Random().nextInt(6) + 5;
    int taille = input_bytes.length + a + b;
    byte[] plaintext_bytes = new byte[taille];
    System.out.println("Input plaintext in hex : " + HexString.fromBytes(input_bytes));
    for(int i = 0; i < taille; i++) {
      if(i < a) {
        plaintext_bytes[i] = (byte)new Random().nextInt(256);
      }else if (i < (input_bytes.length + a) ) {
        plaintext_bytes[i] = input_bytes[i-a];
      }else {
        plaintext_bytes[i] = (byte)new Random().nextInt(256);
      }
    }
    System.out.println("modified plaintext in hex       : " + HexString.fromBytes(
    plaintext_bytes));
    plaintext_bytes = PKCS7.padding(plaintext_bytes, BLOCKSIZE);
    System.out.println("modified pkcs7 plaintext in hex : " + HexString.fromBytes(
    plaintext_bytes));

    CBC cbc = new CBC(this, BLOCKSIZE);
    new Random().nextBytes(init_vect);
    cbc.set_init_vector(init_vect);
    ECB ecb = new ECB(this, BLOCKSIZE);

    int nb_blocks = plaintext_bytes.length / BLOCKSIZE;
    int algo_number = new Random().nextInt(2);
    byte[] result = new byte[nb_blocks * BLOCKSIZE];

    if( algo_number  == 0) {
      System.out.println("Used mode : AES Oracle ECB");
      result = ecb.encrypt(plaintext_bytes);
    }else {
      System.out.println("Used mode : AES Oracle CBC, IV: "+ HexString.fromBytes(
    init_vect));
      result = cbc.encrypt(plaintext_bytes);
    }
    System.out.println("Oracle encryption result in hex : " + HexString.fromBytes(result)
    );
```

```
    return result;
  }
```

Description de ce chiffrement :

- Etape 1 : On génére grace à la classe **"Random"** de Java la clé AES.
- Etape 2 : en génère deux entiers aléatoire a et b compris entre 5 et 10 (nexInt(6) permet de généré un nombre aléatoire entre 0 et 5, le 6 est exclu). La taille du message s'incrémente de (a+b).
- Etape 3 : on génère des **octects aléatoires pour** les **"a" premier octets** et les **"b" derniers octets** du message du message et on met entre les deux le message initial.
- **a et b étant aléatoire** on ne sait pas si le dernier block est complétement rempli => donc on appel le **padding PCKS7** pour le remplir.
- on crée une instance de ECB et CBC (avec vecteur initial aléatoire) puis on génère un nombre aléatoire 0 ou 1 pour choisir lequel utiliser pour le chiffrement.

Detection de ECB/CBC :

- Afin d'être sûre de détecter le mode ECB, il faudra dans le message initial placé une suite de caractères identique assez longue pour pouvoir la voir se répéter dans le chiffrement. ici j'ai placer que des suite de A (car plus facile à visualiser).
- La fonction utilisé pour la détection n'est rien d'autre que celle utilisé dans la question 1.8. Et s'il n'est pas ECB c'est donc forcément CBC.

Resultat:

```
Question 2.3 : An ECB/CBC detection oracle
----- Create AES ORACLE instance-----
----- Use repeating string : AAA...-----
----- ASE Oracle encryption of previous data -----
Random generated Key in hex : d72c8eb6810f6c66c8328503248f288c
Input plaintext in hex          :
41414141414141414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141
modified plaintext in hex       :
6050d2ed3694397b41414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141414141414141d97cf7c401acb3
modified pkcs7 plaintext in hex :
6050d2ed3694397b41414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141414141414141d97cf7c401acb30d0d0d0d0d0d0d0d0d0d0d0d0d0d
Used mode : AES Oracle ECB
Oracle encryption result in hex :
d1087924190b524b0b337d12b1b5f3828092236b36fb7125f0c9023f5375a3058092236b36fb7125f0c9023f
5375a3058092236b36fb7125f0c9023f5375a3058092236b36fb7125f0c9023f5375a3058092236b36fb7125
f0c9023f5375a305b9855d2710fe6af2ee749c22ffd3544b19376c5b0b08e9a6f94bf1d0c69f9aac
----- Guessing Mode : ECB Mode
```

2.4 Byte-at-a-time ECB decryption (Simple)

Pour le choix d'implémentation le message secret est concaténé à **"input_bytes"** l'exterieur lors de l'appel à la méthode **"oracle_ecb"**.

```
public class AES_ORACLE extends AES {
    ...
  public byte[] oracle_ecb(byte[] input_bytes) {
    byte[] plaintext_bytes = input_bytes;
    ECB ecb = new ECB(this, BLOCKSIZE);
    plaintext_bytes = PKCS7.padding(plaintext_bytes, BLOCKSIZE);
    int nb_blocks = plaintext_bytes.length / BLOCKSIZE;
    byte[] result = new byte[nb_blocks * BLOCKSIZE];
    result = ecb.encrypt(plaintext_bytes);
    return result;
  }
```

2.4.1 Detect AES Oracle block size

Pour détecter la taille du block. On commence par encrypté une chaine de caractère à un seul caractère "A" par exemple, puis à chaque itération sur la taille du bloc (**"supposée"**) on rajoute un caractère "A" puis on regarde si le chiffrement à l'itération "n" est retrouvé dans l'itération "n+1".

Car quand cette condition est vérifiée, cela veut dire que le premier bloc a fini (ne changera plus donc) et on va donc passé au prochain.

```
  System.out.println("----- Detect AES Oracle block size -----");
  byte[]   encrypted_bytes_curr = "".getBytes();
  byte[]   encrypted_bytes_prev = "".getBytes();
  int prev_size = 1;
  int block_size;
  String data = "";
  for(block_size = 1; block_size < 24; block_size++) {
    data += "A";
    encrypted_bytes_curr = aes_oracle.oracle_ecb( (data).getBytes() );
    if(Arrays.equals(
        Arrays.copyOf(encrypted_bytes_curr, prev_size), encrypted_bytes_prev)
        ) {
      block_size -= 1;
      break;
    }
    encrypted_bytes_prev= encrypted_bytes_curr;
    prev_size       = encrypted_bytes_prev.length;
  }
  System.out.println("Block size found : " + block_size);
```

Resultat:

```
Question 2.4 : Byte-at-a-time ECB decryption (Simple)
----- Secret String Base64 to String -----
----- Detect AES Oracle block size -----
Block size found : 16
encryp bytes prev    (block_size = 16): eabba67b588bde1ddf86d02ce4917157
encryp bytes current (block_size = 17): eabba67b588bde1ddf86d02ce4917157
0708a366d01f427999467c3967ee8c78
```

Remarque : On remarque ici que à l'iteration 17 : les 16 premier octets (32 digit hexadécimals) de l'iteration 16 se sont répétés.

2.4.2 Detect AES Oracle Mode

```
//Detecter le mode ECB
//faire en sorte d'avoir plusieurs blocks ECB
//puis utiliser la fonction qui detect l'ECB
System.out.println("----- Detect AES Oracle Mode -----");
data = "";
for(int i= 1; i < 64; i++) {
    data += "A";
}
byte[] plaintext_bytes = (data + cipher_secret_string).getBytes();
encrypted_bytes_curr = aes_oracle.oracle_ecb( plaintext_bytes );
System.out.print("Guessing AES Oracle Mode :");
if(ECB.detect_ecb(encrypted_bytes_curr)) {
  System.out.println(" ECB");
}else {
  System.out.println(" not ECB");
}
```

Resultat:

```
----- Detect AES Oracle Mode -----
Guessing AES Oracle Mode : ECB
```

2.4.3 Decrypt Secret String

Ici on nous demande de decrypté la chaine de caractères (variable secret_string), codé en Base64 en utilisant ECB Oracle.
Explication de la méthode : Pour comprendre la méthode à implémenter, on va donné un exemple simple sur un seul block (sachant la taille du bloc ici est de 16 octets).

Etape 1 : Trouvé le premier caractère :
- Si on concidère le message "As", contenant qu'une suite de A.
- On démmare avec 15 A successifs. il manquera un caractère pour former un bloc.
- Si on veut utilisé ECB Oracle, il faudra le concaténé avec notre message secret. Cela veut dire que le dernière caractère manquant au premier bloc n'est rien d'autre que le premier caractère de notre message secret (notre inconnu).
- le message chiffré obtenu dans le premier bloc. va correcspondre au chiffrement de la String "AAA...A ?" (? : notre inconnu).
- On compare ce chiffrement avec les 256 possibilité de " ?". une fois trouvé on aura le premier caractère de notre message secret.

Etape 2 : Trouvé les autre caractères du premier bloc :
- On reduit le nombre de A de 1 (de 15 à 14 ...).
- Au chiffrement oracle ECB, vu que dans l'étape précédent on a détécté un caractère dans le message secret, on va essayer de trouver un second.
- On utilisera le le même raisonnement que la pour le première caractère. Si le caractère trouvé est "X" on aurra à chiffrer maintenant "AAA..AX ?". Encore une fois une nouvelle inconnu à chercher sur les 256 possibilité.

Etape 3 : Trouvé tout les blocs (du message secret) :
- A la fin de la première boucle (Etape 2), on aura trouvé les 16 premiers octect du message secret.

— Dans ce cas, on aura les 15A suivi des 16 octects précédement trouvé "AA...AX_1...X_{16}" ce qui fait 31 caractère connu donc 1 caractère manquant pour remplir un second bloc.
— On recommance l'etape 1 pour trouvé le caractère 17 du message secret. On nous basant sur le deuxième bloc qui est le résultat du chiffrement de "AX_1...X_{16} ?".
— Généralisé ce raisonnement jusqu'arrivé au derniers bloc.

Code :

```
//final question
System.out.println("----- Decrypt Secret String -----");
int    nb_block          = cipher_secret_string.length() / ECB.BLOCKSIZE + 1;
String plaintext_secret_string = "";
for(int blk = 0; blk < nb_block; blk++) {
  for(int byte_index = ECB.BLOCKSIZE-1; byte_index >= 0; byte_index--) {
    String As = "";
    for(int i = 0; i < byte_index; i++) {
      As += "A";
    }
    byte[] cipherAsWithSecret = aes_oracle.oracle_ecb ( ( As + cipher_secret_string ).
  getBytes());
    byte[] cipherAsWithSecret_block = Arrays.copyOfRange( cipherAsWithSecret, blk*ECB.
  BLOCKSIZE, (blk+1)*ECB.BLOCKSIZE);
    for(char c = 0; c < 256; c++ ) {
      String str        = As + plaintext_secret_string + c + cipher_secret_string;
      byte[] cipher     = aes_oracle.oracle_ecb(str.getBytes());
      byte[] cipher_blk = Arrays.copyOfRange(cipher, blk*ECB.BLOCKSIZE, (blk+1)*ECB.
  BLOCKSIZE);
      if(Arrays.equals(cipherAsWithSecret_block, cipher_blk)) {
        plaintext_secret_string += c;
        break;
      }
    }
  }
}
System.out.println(plaintext_secret_string);
```

Resultat:

```
----- Decrypt Secret String -----
Rollin' in my 5.0
With my rag-top down so my hair can blow
The girlies on standby waving just to say hi
Did you stop? No, I just drove by
```