INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Java

benjamin.nguyen@insa-cvl.fr

Chiffrement en Java

“It is a secret in the Oxford sense: you may tell it to only one person at a time.” – Lord Franks, 1977

But du cours (compétences)

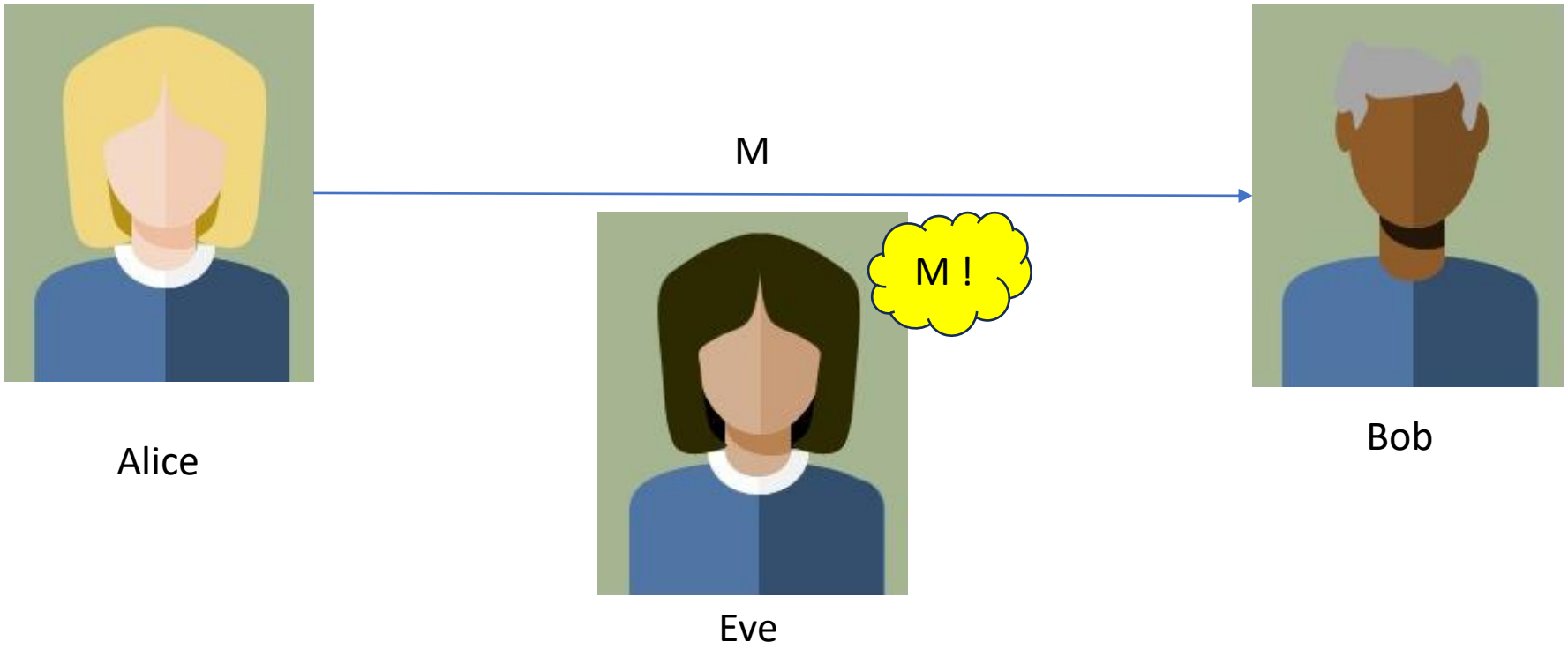
- Savoir coder une application complexe en Java (avec des tests unitaires)
- Savoir débugger une application complexe en Java
- Savoir gérer les fichiers en Java
- Savoir gérer la persistence en Java avec la sérialisation
- Savoir utiliser les threads en Java
- Savoir créer une interface graphique simple en Java (approche modèle MVC)

Si le temps le permet …

- Connaître les bibliothèques cryptographiques en Java
- Comprendre et avoir utilisé le mécanisme d'introspection en Java
- Savoir gérer la persistence en Java avec une base de données

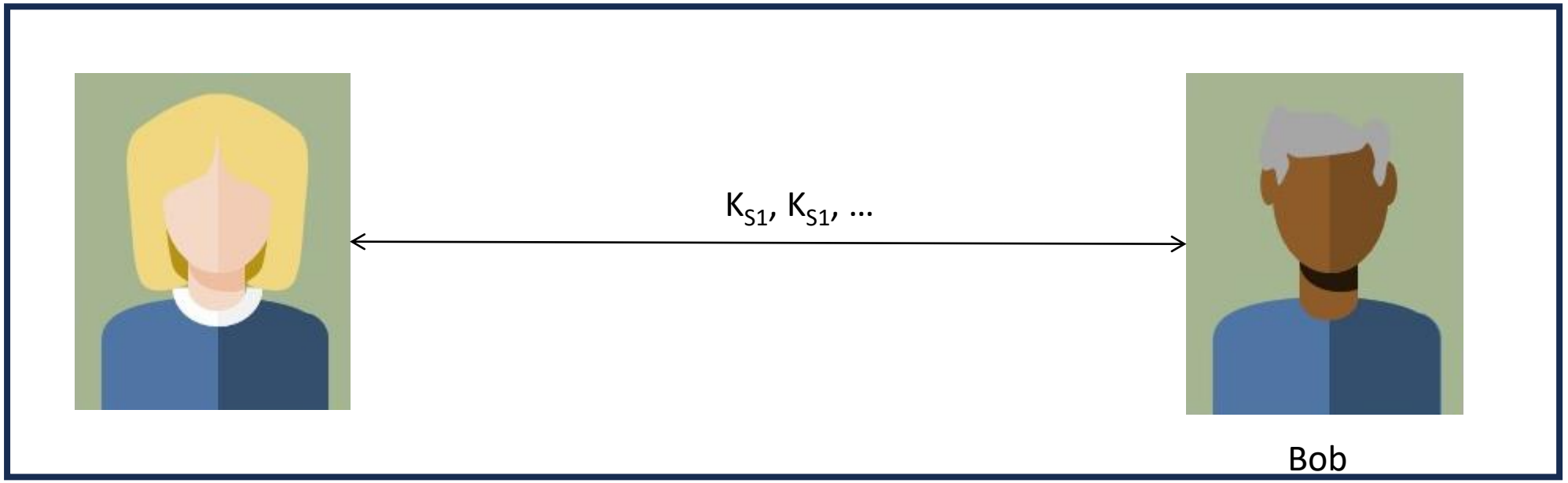
Eléments de cryptographie “de base”

Alice veut transmettre une information secrete (M) à Bob sur un canal de transmission qui n’est pas de confiance.



Quelles hypothèses ? Cryptographie symmétrique

- Alice et Bob se sont mis d'accord sur un protocole (public)
- Alice et Bob se sont mis d'accord sur une (ou plusieurs) clé secrêtes K_S de manière *privée*.



Pourquoi symmétrique ?

- Alice et Bob partagent une même clé (ou ensemble de clés).

Vocabulaire

- Chiffrer et déchiffrer : quand on connait l’algorithme
- Encrypt et decrypt : en anglais
- Crypter et décrypter : non !

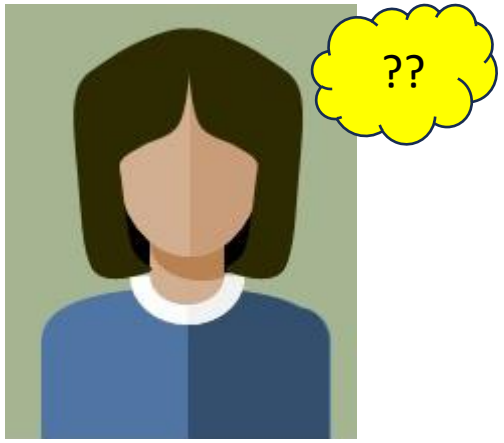
Exemple : chiffrement de Jules César (chiffrement par décalage)

- Protocole :
  - Chiffrer : Décaler les lettres de l'alphabet d'une position *p* vers la fin de l'alphabet.
  - Déchiffrer : Décaler les lettres d'une position *p* vers le début de l'alphabet.
- Clé :
  - Chiffrement : $K_S=p$
  - Déchiffrement : $K_S=p$

M = alea jacta est
K_S=3
C = dohd mdfwd hvw

Exemple

(Cesar, S=dohd mdfwd hvw)



Alice

M = alea jacta est
K = 3
$Cesar_C(M) = S$ = dohd mdfwd hvw

Eve

Peut faire des attaques
e.g. attaque exhaustive
vu la taille de la clé

Bob

K = 3
S = dohd mdfwd hvw
$Cesar_D(S) = M$ = alea jacta est

Divers types d'attaque

- Dépend de la connaissance de l'attaquant (il connait toujours l'algo)
  - On connait S et on veut retrouver M
  - On connait S et on veut retrouver K_S
  - On connait S_1, S_2, … S_j et on veut retrouver M_1 (COA Cyphertext Only Attack)
  - On connait S_1, S_2, … S_j et on connait M_1, M_2, … M_{j-1} et on veut retrouver M_j (KPA Known Plaintext Attacks)
  - On peut choisir M_1, on nous donne S_1 et on doit déchiffrer S (CPA Chosen Plaintext Attacks)
  - On peut choisir S_1, on nous donne M_1 et on doit trouver K_S (CCA Chosen Cyphertext Attacks)

Algorithmes de chiffrement modernes

- On ne considèrera ici que AES (Rijndael)
  - Utilisé depuis 2001 et standard officiel depuis 2002
  - Auteurs : Joan Daemen and Vincent Rijmen → Rijndael
  - Ont gagné la compétition NIST ‘AES’ (Advances Encryption Standard)
  - Blocs de 128 bits blocks avec des clés de 128, 192 ou 256 bits
  - Rapide et nécessite peu de mémoire !
- Pas de preuve ou de publications à l’heure actuelle qui montrent de faiblesses pour AES

Code exact d'AES ? Voir cours de cryptographie

- On considère ici que c'est une boîte noire
- Entrée :
  - M = Un *bloc* de 128 bits (potentiellement *paddés*) soit 16 octets, placés dans une matrice 4x4
  - K_S= 128, 192 ou 256 bits qui sert à dériver des clés intermédiaires
  - Si on utilise le mode CBC, il faut aussi un vecteur d'initialisation (IV)

Code d'AES vu de loin

Source Wikipedia AES

The key schedule [edit]

Define:

- N as the length of the key in 32-bit words: 4 words for AES-128, 6 words for AES-192, and 8 words for AES-256
- $K_0, K_1, \ldots K_{N-1}$ as the 32-bit words of the original key
- R as the number of round keys needed: 11 round keys for AES-128, 13 keys for AES-192, and 15 keys for AES-256[note 4]
- $W_0, W_1, \ldots W_{4R-1}$ as the 32-bit words of the expanded key[note 5]

Also define RotWord as a one-byte left circular shift:[note 6]

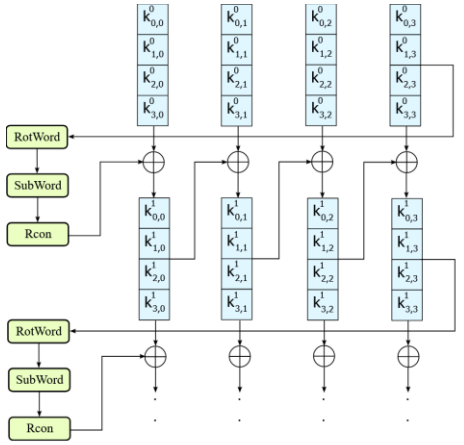
$$\mathrm{RotWord}([\,b_0 \quad b_1 \quad b_2 \quad b_3\,]) = [\,b_1 \quad b_2 \quad b_3 \quad b_0\,]$$

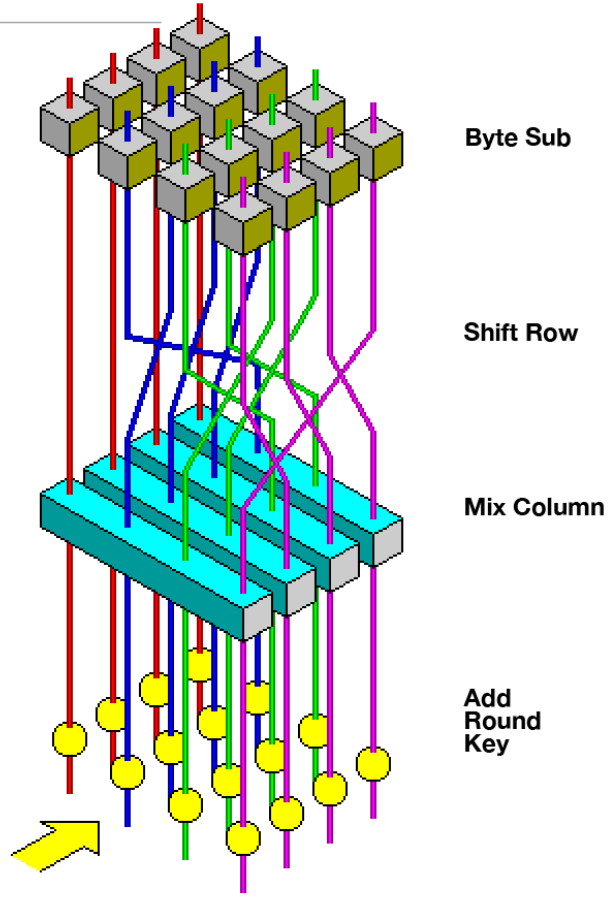
and SubWord as an application of the AES S-box to each of the four bytes of the word:

$$\mathrm{SubWord}([\,b_0 \quad b_1 \quad b_2 \quad b_3\,]) = [\,\mathrm{S}(b_0) \quad \mathrm{S}(b_1) \quad \mathrm{S}(b_2) \quad \mathrm{S}(b_3)\,]$$

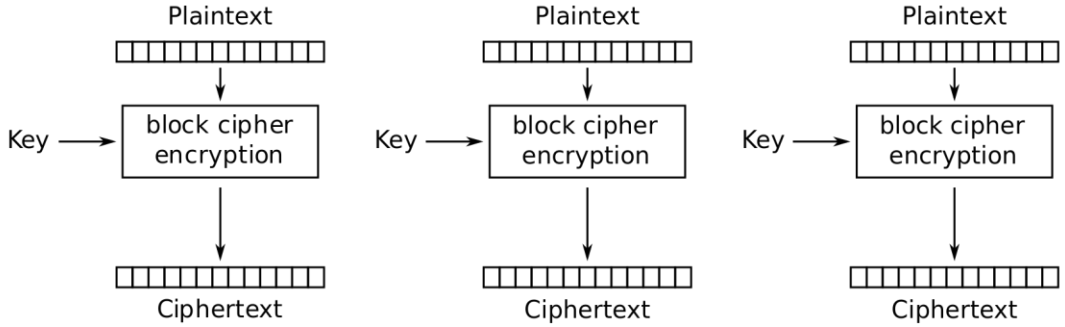
Then for $i = 0 \ldots 4R-1$:

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \mathrm{SubWord}(\mathrm{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod N \\ W_{i-N} \oplus \mathrm{SubWord}(W_{i-1}) & \text{if } i \geq N,\ N > 6, \text{ and } i \equiv 4 \pmod N \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

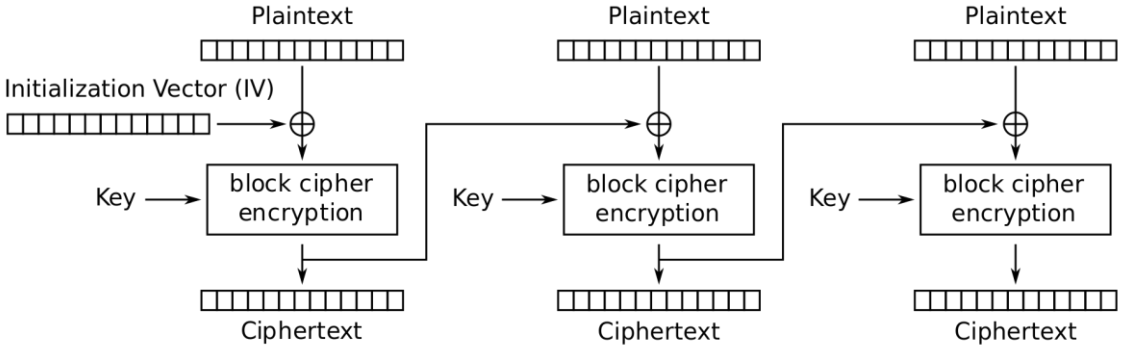




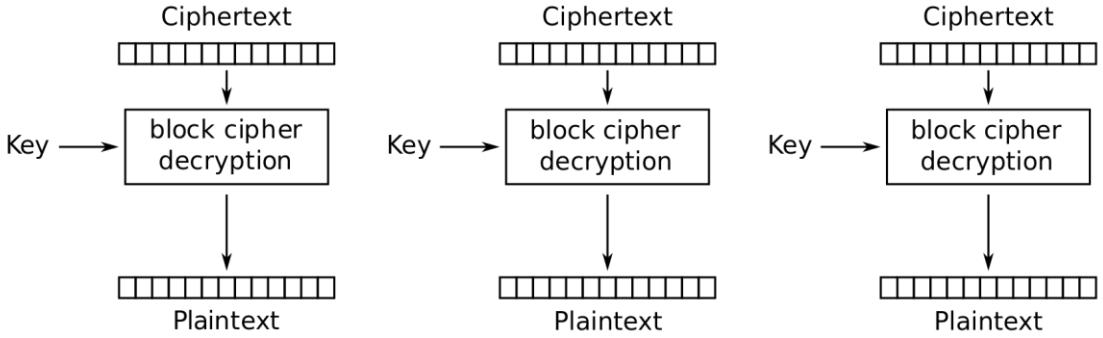
ECB vs CBC



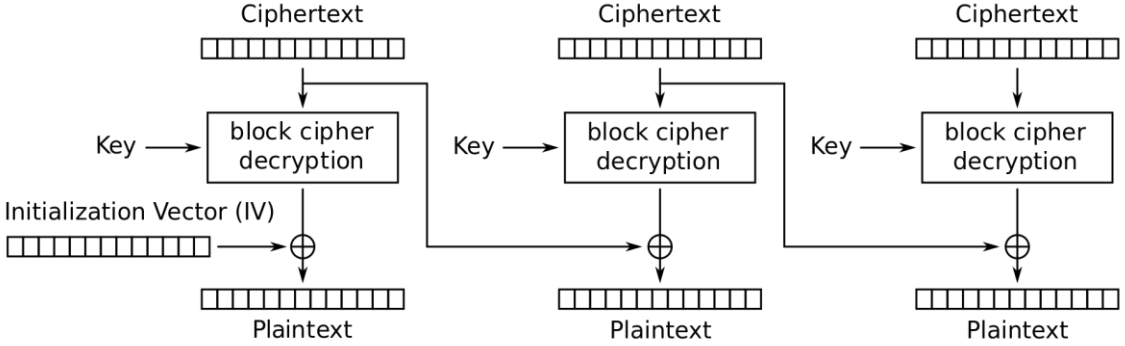
Electronic Codebook (ECB) mode encryption



Cipher Block Chaining (CBC) mode encryption



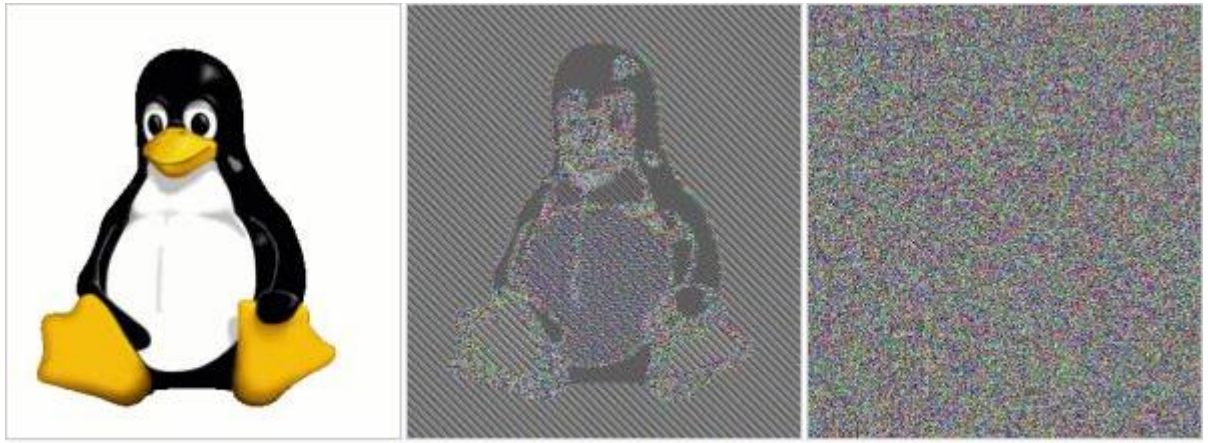
Electronic Codebook (ECB) mode decryption



Cipher Block Chaining (CBC) mode decryption

Source : Wikipedia

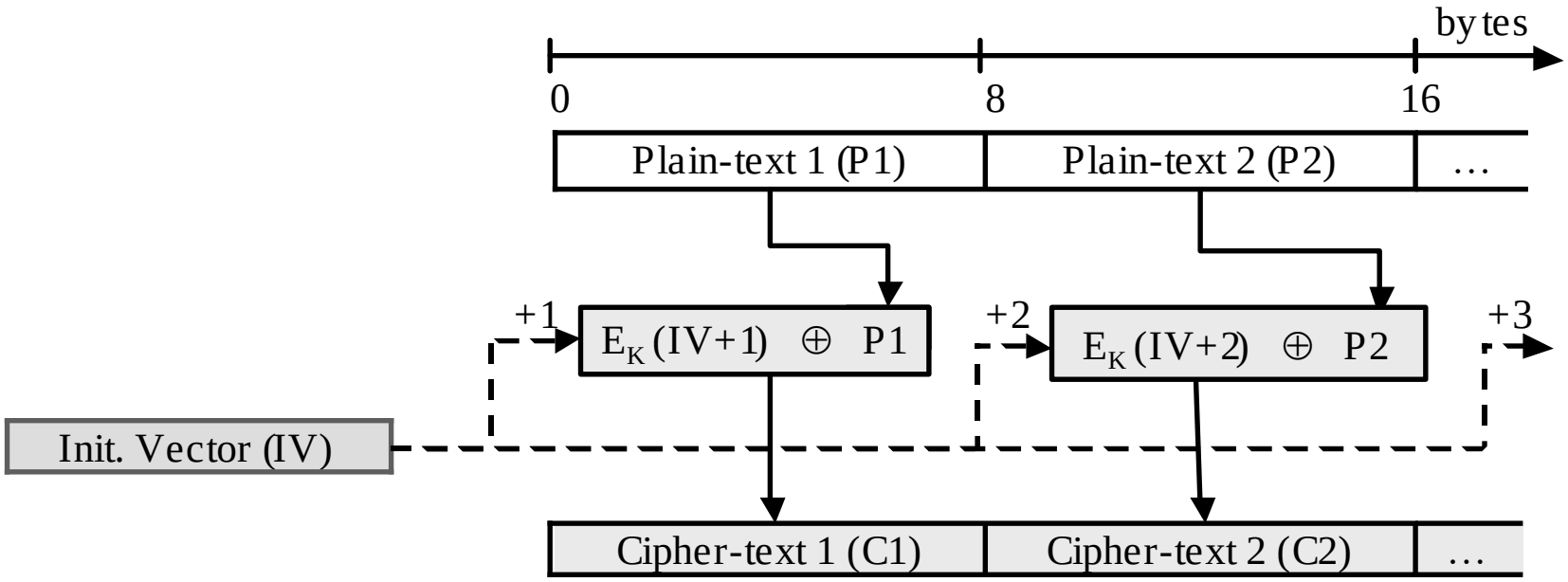
Inconvénient de CBC : vitesse
Inconvénient de ECB : sécurité



Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

Un certain degré de parallélisme lors du déchiffrement est possible : la partie hors XOR.

Mode Counter



Et en Java ??

- Il faut :
  - Pouvoir générer une clé AES (à partir d'informations fournies par l'utilisateur)
  - Pouvoir chiffrer (en choisissant mode ECB, CBC, etc.)
  - Pouvoir déchiffrer

Génération d’une clé

1) Sans interaction avec un utilisateur :

Classes `javax.crypto.KeyGenerator` et `javax.crypto.SecretKey`

```
public static SecretKey generateKey(int n) throws NoSuchAlgorithmException {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");   // algo
        keyGenerator.init(n);    // taille de la clé (128, 192, 256 pour AES)
        SecretKey key = keyGenerator.generateKey();
        return key;
        }
```

Génération d'une clé

2) Avec interaction avec un utilisateur (mot de passe)

On utilise une *fonction de dérivation de clé* (e.g. Password Based Key Derivation Function PBKDF, e.g. `PBKDF2WithHmacSHA256`)

Classes `javax.crypto.SecretKeyFactory` et `javax.crypto.SecretKey` (des checks faits avec `javax.crypto.spec.PBEKeySpec`)

```
public static SecretKey getKeyFromPassword(String password, String salt)
   throws NoSuchAlgorithmException, InvalidKeySpecException {

          SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
          KeySpec spec = new PBEKeySpec(password.toCharArray(), salt.getBytes(), 42, 256);
          SecretKey secret = new SecretKeySpec(factory.generateSecret(spec).getEncoded(), "AES");
          return secret;
          }
```

- Note : le 3e paramètre (iteration count) indique le nombre de fois qu'on hash le mot de passe (pour se protéger contre des rainbow tables)

Génération du vecteur d'initialisation

`javax.crypto.spec.IvParameterSpec`

- On le génère à l'aide d'un générateur de nombres aléatoires avec `java.security.SecureRandom`

```
public static IvParameterSpec generateIv() {
    byte[] iv = new byte[16];
    new SecureRandom().nextBytes(iv);
    return new IvParameterSpec(iv);
}
```

Attention à du code du genre :

```
public static IvParameterSpec zeroIv() {
    byte[] iv = new byte[16];
    return new IvParameterSpec(iv);
}
```

Chiffrement d’une chaîne de caractères

javax.crypto.Cipher

```
public static String encrypt(String algorithm, String input, SecretKey key,
    IvParameterSpec iv) throws NoSuchPaddingException, NoSuchAlgorithmException,
    InvalidAlgorithmParameterException, InvalidKeyException,
    BadPaddingException, IllegalBlockSizeException {

    Cipher cipher = Cipher.getInstance(algorithm);
    cipher.init(Cipher.ENCRYPT_MODE, key, iv);
    byte[] cipherText = cipher.doFinal(input.getBytes());
    return Base64.getEncoder().encodeToString(cipherText);
}
```

Types d’algorithmes supportés :

- AES/CBC/PKCS5Padding

Noms des algos

Algorithm Name	Description
AES	Advanced Encryption Standard as specified by NIST in FIPS 197. Also known as the Rijndael algorithm by Joan Daemen and Vincent Rijmen, AES is a 128-bit block cipher supporting keys of 128, 192, and 256 bits. To use the AES cipher with only one valid key size, use the format AES_<n>, where <n> can be 128, 192 or 256.
AESWrap	The AES key wrapping algorithm as described in RFC 3394. To use the AESWrap cipher with only one valid key size, use the format AESWrap_<n>, where <n> can be 128, 192, or 256.
ARCFOUR	A stream cipher believed to be fully interoperable with the RC4 cipher developed by Ron Rivest. For more information, see K. Kaukonen and R. Thayer, "A Stream Cipher Encryption Algorithm 'Arcfour'", Internet Draft (expired).
Blowfish	The Blowfish block cipher designed by Bruce Schneier.
ChaCha20	The ChaCha20 stream cipher as defined in RFC 7539.
ChaCha20-Poly1305	The ChaCha20 cipher in AEAD mode using the Poly1305 authenticator, as defined in RFC 7539.
DES	The Digital Encryption Standard as described in FIPS PUB 46-3.
DESede	Triple DES Encryption (also known as DES-EDE, 3DES, or Triple-DES). Data is encrypted using the DES algorithm three separate times. It is first encrypted using the first subkey, then decrypted with the second subkey, and encrypted with the third subkey.
DESedeWrap	The DESede key wrapping algorithm as described in RFC 3217.
ECIES	Elliptic Curve Integrated Encryption Scheme
PBEWith<digest>And<encryption> PBEWith<prf>And<encryption>	The password-based encryption algorithm defined in PKCS #5, using the specified message digest (<digest>) or pseudo-random function (<prf>) and encryption algorithm (<encryption>). Examples: **PBEWithMD5AndDES**: The PBES1 password-based encryption algorithm as defined in PKCS #5: Password-Based Cryptography Specification, Version 2.1. Note that this algorithm implies CBC as the cipher mode and PKCS5Padding as the padding scheme and cannot be used with any other cipher modes or padding schemes. **PBEWithHmacSHA256AndAES_128**: The PBES2 password-based encryption algorithm as defined in PKCS #5: Password-Based Cryptography Specification, Version 2.1.
RC2	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc.
RC4	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc. (See note prior for ARCFOUR.)
RC5	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc.
RSA	The RSA encryption algorithm as defined in PKCS #1 v2.2

Mode de chiffrement

Algorithm Name	Description
NONE	No mode.
CBC	Cipher Block Chaining Mode, as defined in FIPS PUB 81.
CCM	Counter/CBC Mode, as defined in NIST Special Publication SP 800-38C.
CFB, CFBx	Cipher Feedback Mode, as defined in FIPS PUB 81. Using modes such as CFB and OFB, block ciphers can encrypt data in units smaller than the cipher's actual block size. When requesting such a mode, you may optionally specify the number of bits to be processed at a time by appending this number to the mode name as shown in the "*DES/CFB8/NoPadding*" and "*DES/OFB32/PKCS5Padding*" transformations. If no such number is specified, a provider-specific default is used. (For example, the SunJCE provider uses a default of 64 bits for DES.) Thus, block ciphers can be turned into byte-oriented stream ciphers by using an 8-bit mode such as CFB8 or OFB8.
CTR	A simplification of OFB, Counter mode updates the input block as a counter.
CTS	Cipher Text Stealing, as described in Bruce Schneier's book *Applied Cryptography-Second Edition*, John Wiley and Sons, 1996.
ECB	Electronic Codebook Mode, as defined in FIPS PUB 81 (generally this mode should not be used for multiple blocks of data).
GCM	Galois/Counter Mode, as defined in NIST Special Publication SP 800-38D.
OFB, OFBx	Output Feedback Mode, as defined in FIPS PUB 81. Using modes such as CFB and OFB, block ciphers can encrypt data in units smaller than the cipher's actual block size. When requesting such a mode, you may optionally specify the number of bits to be processed at a time by appending this number to the mode name as shown in the "*DES/CFB8/NoPadding*" and "*DES/OFB32/PKCS5Padding*" transformations. If no such number is specified, a provider-specific default is used. (For example, the SunJCE provider uses a default of 64 bits for DES.) Thus, block ciphers can be turned into byte-oriented stream ciphers by using an 8-bit mode such as CFB8 or OFB8.
PCBC	Propagating Cipher Block Chaining, as defined by Kerberos V4.

Mode de padding

Algorithm Name	Description
NoPadding	No padding.
ISO10126Padding	This padding for block ciphers is described in 5.2 Block Encryption Algorithms in the W3C “XML Encryption Syntax and Processing” document.
OAEPPadding, OAEPWith<digest>And<mgf>Padding	Optimal Asymmetric Encryption. Padding scheme defined in PKCS #1, where <digest> should be replaced by the message digest and <mgf> by the mask generation function. Examples: **OAEPWithMD5AndMGF1Padding** and **OAEPWithSHA-512AndMGF1Padding**. If OAEPPadding is used, Cipher objects are initialized with a javax.crypto.spec.OAEPParameterSpec object to supply values needed for OAEPPadding.
PKCS1Padding	The padding scheme described in PKCS #1 v2.2, used with the RSA algorithm.
PKCS5Padding	The padding scheme described in PKCS #5: Password-Based Cryptography Specification, version 2.1.
SSL3Padding	The padding scheme defined in the SSL Protocol Version 3.0, November 18, 1996, section 5.2.3.2 (CBC block cipher): block-ciphered struct { opaque content[SSLCompressed.length]; opaque MAC[CipherSpec.hash_size]; uint8 padding[GenericBlockCipher.padding_length]; uint8 padding_length; } GenericBlockCipher; The size of an instance of a GenericBlockCipher must be a multiple of the block cipher’s block length. The padding length, which is always present, contributes to the padding, which implies that if: sizeof(content) + sizeof(MAC) % block_length = 0, padding has to be (block_length - 1) bytes long, because of the existence of padding_length. This makes the padding scheme similar (but not quite) to PKCS5Padding, where the padding length is encoded in the padding (and ranges from 1 to block_length). With the SSL scheme, the sizeof(padding) is encoded in the always present padding_length and therefore ranges from 0 to block_length-1.

Chiffrement avec RSA

- A voir en TD !