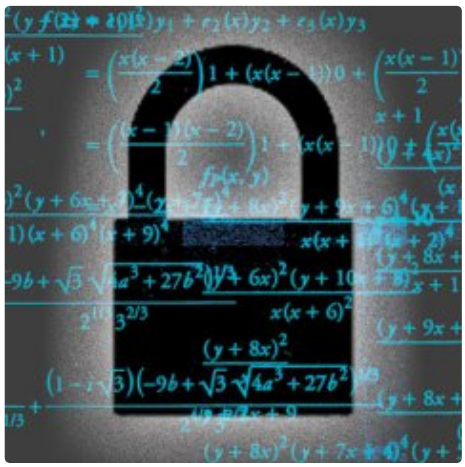
534 days ago

Chiffrer/déchiffrer simplement avec AES en JAVA

Le module de cryptographie (*javax.crypto*) de Java permet facilement le (dé)chiffrement d'un document ou d'une simple chaîne de caractères :

- soit en utilisant une même clé pour chiffer/déchiffrer (chiffrement symétrique) : l'algorithme principalement utilisé est AES (Advanced Encryption Standard) , il nous permet d'utiliser une clé de taille 16, 24 ou 32 caractères (soit 128, 192 ou 256 bits).

- soit en utilisant des clés différentes (chiffrement asymétrique), une clé publique et une clé privée : l'algorithme utilisé est RSA .



Plus la taille de votre clé est longue, plus il sera difficile de déchiffrer vos documents, cependant voici ce que dit le gouvernement américain (§6) à ce propos :

> *L'architecture et la longueur de toutes les tailles de clés de l'algorithme AES (128, 192 et 256) sont suffisantes pour protéger des documents*

> *classifiés jusqu'au niveau « SECRET ». Le niveau « TOP SECRET » nécessite des clés de 192 ou 256 bits. L'implémentation de l'AES dans des produits destinés à la protection des systèmes et/ou documents liés à la sécurité nationale doit faire l'objet d'une analyse et d'une certification par la NSA avant leur acquisition et leur utilisation*

Chiffrement "SECRET"

Pour un chiffrement de niveau "SECRET", voici un exemple de classe utilitaire permettant rapidement de (dé)chiffrer avec l'algorithme AES cité ci-dessus :

```
import com.google.common.base.Charsets;
import org.apache.commons.codec.binary.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Date;

@Component
public class CipherUtilSecret {

    private final Logger log = LoggerFactory.getLogger(CipherUtilSecret.class

    public static final String CIPHER_ALGORITHM = "AES";
    public static final String KEY_ALGORITHM = "AES";
    public static final byte[] SECRET_KEY = "16BYTESSECRETKEY" .getBytes(Char

    public String decrypt(String encryptedInput) {
        try {
            Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
            cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(SECRET_KEY, KE
            return new String(cipher.doFinal(Base64.decodeBase64(encryptedInp

        } catch (Exception e) {
            log.warn(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
```

```
    public String encrypt(String str) {
        try {
            Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(SECRET_KEY, KE
            return Base64.encodeBase64URLSafeString(cipher.doFinal(str.getByt

        } catch (Exception e) {
            log.warn(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) {
        CipherUtilSecret cipherUtil = new CipherUtilSecret();
        // Encryption
        String encryptedString = cipherUtil.encrypt("1,2,3 allons dans les bo
        // Before Decryption
        System.out.println("Avant déchiffrement : " + encryptedString);
        String s = cipherUtil.decrypt(encryptedString);
        System.out.println("Après déchiffrement : " + s);
    }
}
```

Attention à bien utiliser une clé de chiffrement d'exactement 16 caractères sinon vous allez générer des exceptions de ce type :

```
Caused by: java.security.InvalidKeyException: Illegal key size or default par
```

Chiffrement "TOP SECRET"

Pour un chiffrement de type "TOP SECRET", il vous faut au préalable installer la Java Cryptography Extension (JCE) correspondant à votre version de Java installé sur votre machine : Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8 Download .
Cette extension va nous permettre de dépasser la limite standard des 16 caractères pour la clé de chiffrement.

AES nous autorisant à utiliser une clé de taille 16, 24 ou 32 caractères, nous pouvons procéder de deux façons :

- soit nous utilisons une clé de (dé)chiffrement avec exactement une taille de 24 ou 32 caractères
- soit nous générons une clé de (dé)chiffrement à partir d'une "passphrase" en utilisant la méthode de hashage SHA-256 (cela génèrera une chaîne de caractères d'exactement 32 caractères).

Voici un exemple de classe utilitaire permettant de (dé)chiffrer avec l'algorithme AES en utilisant la deuxième option :

```
import com.google.common.base.Charsets;
import org.apache.commons.codec.binary.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

@Component
public class CipherUtilTopSecret {

    private final Logger log = LoggerFactory.getLogger(CipherUtilTopSecret.cl

    public static final String CIPHER_ALGORITHM = "AES";
    public static final String KEY_ALGORITHM = "AES";
    public static final String PASS_HASH_ALGORITHM = "SHA-256";
    public static final String DEFAULT_PASS = "Your Default Security PassPhra

    public String encrypt(String data) {
        try {
            Cipher cipher = buildCipher(DEFAULT_PASS, Cipher.ENCRYPT_MODE);
            byte[] dataToSend = data.getBytes(Charsets.UTF_8);
            byte[] encryptedData = cipher.doFinal(dataToSend);
            return Base64.encodeBase64URLSafeString(encryptedData);
```

```
        } catch (Exception e) {
            log.warn(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }

    public String decrypt(String encryptedValue) {
        try {
            Cipher cipher = buildCipher(DEFAULT_PASS, Cipher.DECRYPT_MODE);
            byte[] encryptedData = Base64.decodeBase64(encryptedValue);
            byte[] data = cipher.doFinal(encryptedData);
            return new String(data, Charsets.UTF_8);

        } catch (Exception e) {
            log.warn(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }

    private Cipher buildCipher(String password, int mode) throws NoSuchAlgori
        Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
        Key key = buildKey(password);
        cipher.init(mode, key);
        return cipher;
    }

    private Key buildKey(String password) throws NoSuchAlgorithmException, Un
        MessageDigest digester = MessageDigest.getInstance(PASS_HASH_ALGORITH
        digester.update(String.valueOf(password).getBytes(Charsets.UTF_8.name
        byte[] key = digester.digest();
        return new SecretKeySpec(key, KEY_ALGORITHM);
    }


    public static void main(String[] args) {
        CipherUtilTopSecret cipherUtil = new CipherUtilTopSecret();
        // Encryption
        String encryptedString = cipherUtil.encrypt("4,5,6 cueillir des ceris
        // Before Decryption
        System.out.println("Avant decrypt : " + encryptedString);
        String s = cipherUtil.decrypt(encryptedString);
        System.out.println("Après decrypt : " + s);
    }

}
```