1.TripleDES Implementation

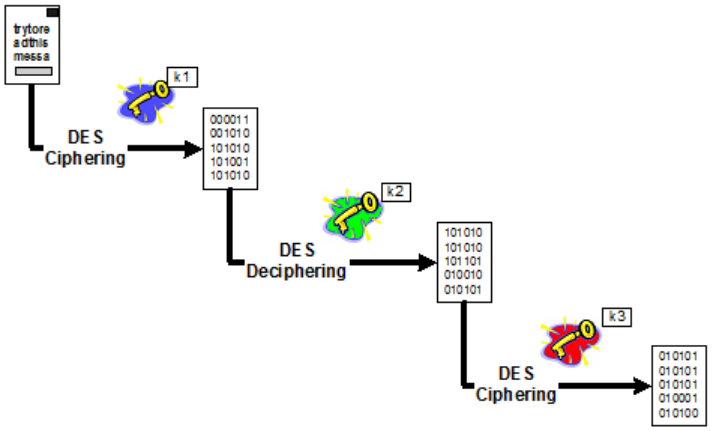
Objective:

The goal of the exercise is to get familiar with the API of javax.crypto. In order to so, you will have to implement 3DES in EBC and CBC mode.

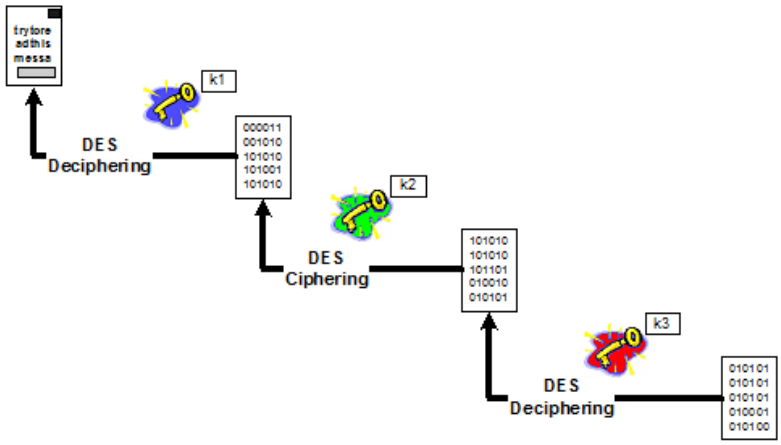
You will have to use the

1. Reminder

3DES is based on the symmetric algorithm **DES**.
3DES Encryption is based on the following schema:



Decryption is based on the following schema :



2. DES encryption

Generate 3 DES keys

[*javax.crypto.KeyGenerator*] and [*javax.crypto.SecretKey*]

1. Generate 3 DES keys and store them into the following files: *DESKey1*, *DESKey2*, *DESKey3*.

Hint: *look at javax.crypto.KeyGenerator and its methods KeyGenerator ::getInstance(String algorithm) et KeyGenerator::generateKey(). The algorithm to be used here is « DES ».*

In EBC mode

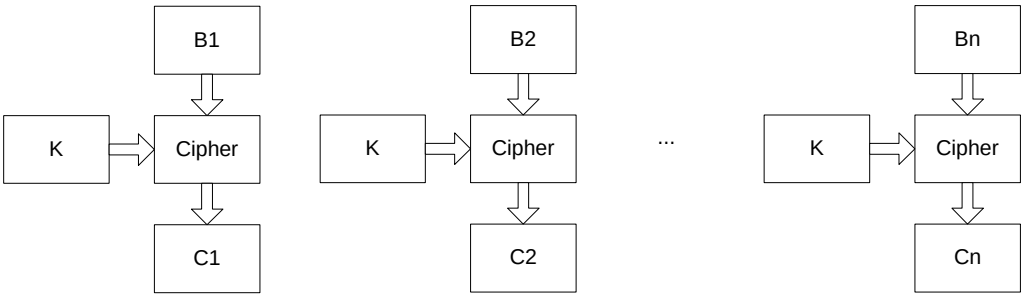


Figure 1: EBC Block Cipher

Ciphering - **[***javax.crypto.Cipher***]**

1. DES ciphering with the first key
2. DES deciphering with the second key
3. DES ciphering with the third key

Deciphering- [*javax.crypto.Cipher*]

1. DES decipehering with the thrid key
2. DES cipehring with the second key
3. DES deciphering with the first key

Hint:
Cipher Name is "DES" By default, "DES" implements DES/EBC.
Use NoPadding as padding mechanism.

3. In CBC mode

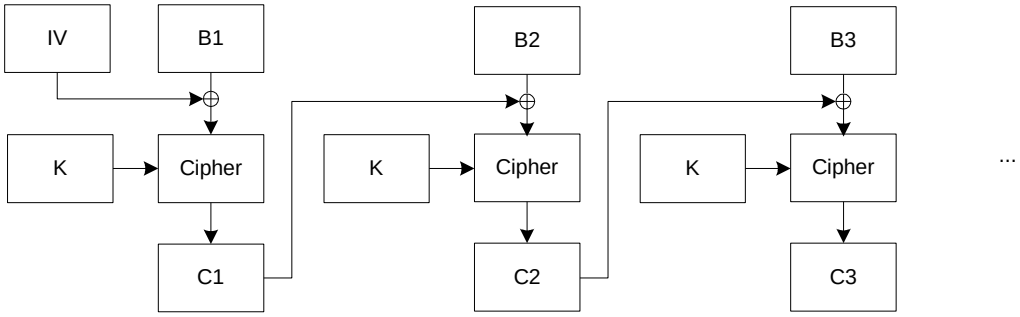


Figure 2: CBC Block Cipher

Ciphering
[*javax.crypto.Cipher***] et [***javax.crypto. AlgorithmParameterSpec***]**

1. Create anInitialisation Vector
2. DES Ciphering with the first key
3. DES deciphering with the second key
4. DES ciphering with the third key

Deciphering
[*javax.crypto.Cipher***]**

1. Reuse the SAME IV
2. DES deciphering with the third key
3. DES ciphering with the second key
4. DES deciphering with the first key

Hint :
Cipher engine can be initialized with an object of type javax.crypto.AlgorithmParameterSpec.
Use NoPadding.

2. RSA Signature Implementation

Objective:

The goal of the exercise is to get familiar with the API of java.security. In order to so, you will have to implement RSA signature and encryption.

1. Generation of a public/private key pair

[java.security.KeyPairgenerator]

In method *Entity::Entity()*

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

2. RSA Signature

Signature[java.security.Signature]

In method *Entity::sign()*
Create an signature object java.security.signature for « MD5withRSA ».
Initialise the object with the private key in SIGN_MODE.
Sign

Check signature [java.security.Signature]

In method *Entity::checkSignature()*
Create an objet java.security.Signature
Initialise it in VERIFY_MODE mode with the public key
Check the signature.

3. Implementation of your own RSA signature

Signature

In methode *Entity::mySign()*
Implement your own signature using

- javax.crypto.Cipher with RSA in ENCRYPT_MODE mode
- java.security.MessageDigest with MD5.

Check signature

In methode *Entity::myCheckSignature()*
Implement your own signature verification using

- javax.crypto.Cipher with RSA in DECRYPT_MODE mode
- java.security.MessageDigest with MD5

4. RSA Ciphering

Warning : RSA implementation by SUN does not support message greater than 127 bytes.

RSAEncryption

In method *Entity::encrypt()*
Use method *javax.crypto.Cipher::doFinal()*

RSADecryption

In method *Entity::decrypt()*.
Use method *javax.crypto.Cipher::doFinal()*

3. Secure session key exchange

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 generate 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.

You can also refer to slide 88 to 92 from application security lecture for further details on this secure session key exchange.