Application Security – Lab Session 1
Java Security

laurent.gomez@sap.com

Instructions

In this lab session, you will have to answer questions in this document, and complete two java projects: 3DES and RSA.

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

This document has to be completed, zip it together with your JAVA projects and send it to laurent.gomez@sap.com as follows:

- Zip it and rename it to **LabSession1-FirstNameStudent1-FirstNameStudent2.ZIP**
- Send it to laurent.gomez@sap.com with following subject: **[Application Security Polytech] – LabSession1**
- Put in CC the second student if any.

This lab session will be evaluated on 20, and will count for 25% of the final Application Security grade.

Application Security – Lab Session 1
Java Security

laurent.gomez@sap.com

3DES

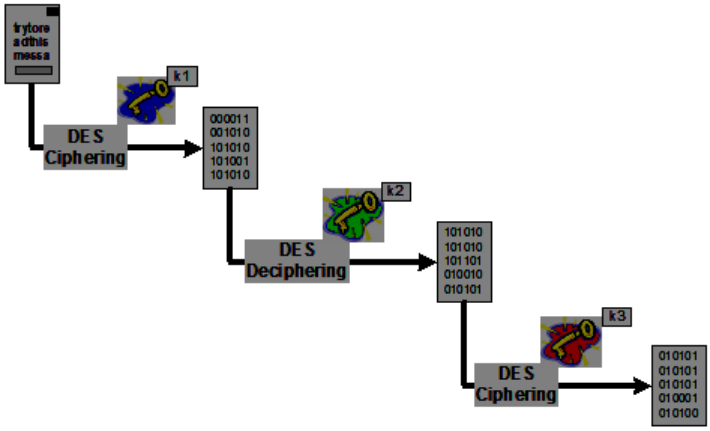
(10 points)

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

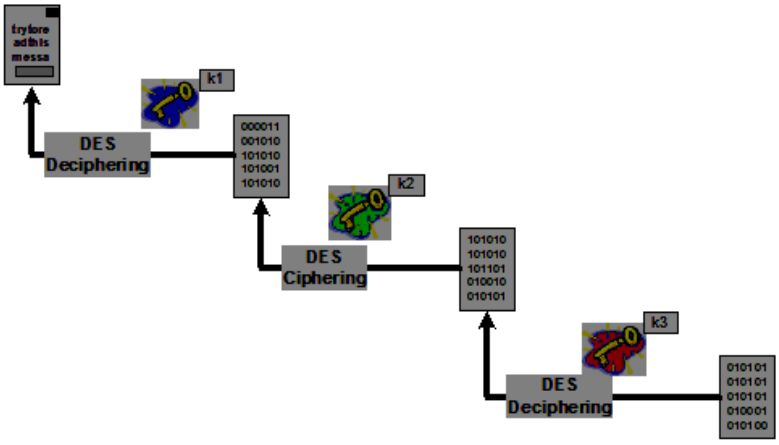
Background information

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

3DES encryption is performed as depicted in the following figure:



3DES decryption is performed as depicted in the following figure:



Several keying configuration are possible with 3DES.
In this lab session, K1, K2 and K3 will be independent.

Application Security – Lab Session 1
Java Security

laurent.gomez@sap.com

1. Data Encryption Standard (1 point)

What is the size of DES cipher key's size ? (0.5 point)
A) DES has a 64-bit key size, but only 56 of those are used during encryption. The other 8 are "parity bits". The parity bits are used for error detection during key storage, distribution etc.

What are the size of the cipher blocks ? (0.5 point)
A) The size of the cipher block in DES is 64 bits.

laurent.gomez@sap.com

2. DES/CBC/NoPadding (4,5 points)

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

Cipher Block Chaining (CBC) encryption improves the security of the Electronic Code Block mode (ECB/EBC).
In fact, ECB transform equal blocks of plaintext in equal blocks of ciphertext, simplifying the cryptographic analysis of the ciphertext.
CBC (Figure 1) solves the issue making the transformation of each block dependent on the result of the previous block. In detail the input of an encryption step is the result of a XOR operation of the plaintext and the output of the previous step. The first block is combined with the initialization vector, provided as parameter to the algorithm.

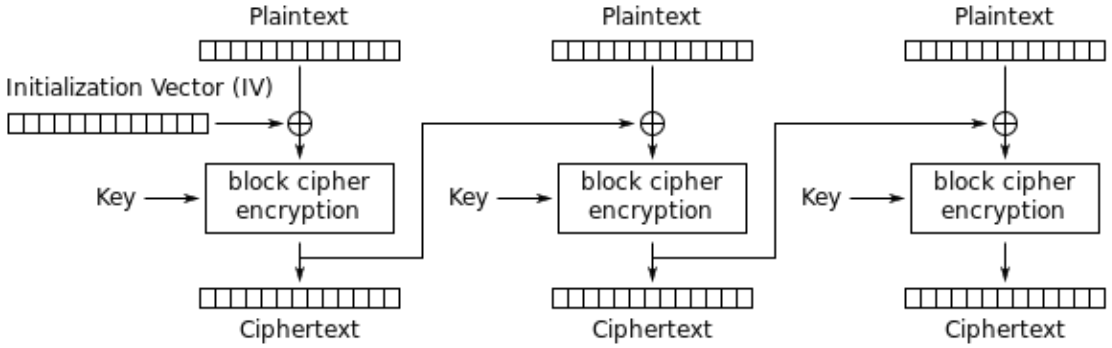


Figure 1 - Cipher Block Chaining mode encryption diagram

The initialization vector is not a secret and hence can be transmitted in plaintext to the decoder who will use the initialization vector and the secret key on the cipher text to decode it.

2.2. Explain NoPadding (0,5 point)
Block ciphers needs that the length of the plaintext is a multiple of the size of a single block. If the plaintext alone has not the required size, a padding is added at the end.
The formula to determine the number of padding bits "p" to be added is: **$p = k - (m \bmod k)$** where k is the block size in bits, and m is the size of the plaintext in bits
Usually this operation is performed by the algorithm, as the padding is provided as parameter by the developer.
In some cases, the developer would like to take the responsibility of providing the plaintext of the correct length, adding the padding of his own if necessary so is no longer necessary that the algorithm will perform that operation, thus NoPadding parameter is used. The algorithm will just check that the size of the input is correct, otherwise it will raise an exception.

For the next two questions, you are asked to encrypt the following file *`./ebc/clearTextFileEBC`*

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

laurent.gomez@sap.com

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

laurent.gomez@sap.com

2. DES/EBC/NoPadding

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

EBC (Electronic Book Code) can parallelize the encryption, while CBC can't. Moreover, it doesn't need an initialization vector. This is because, as we can see comparing figure 1 and 2, EBC only depends on a section of the plaintext, while CBC needs to wait that the previous block has been encrypted or a IV (Initialization Vector).

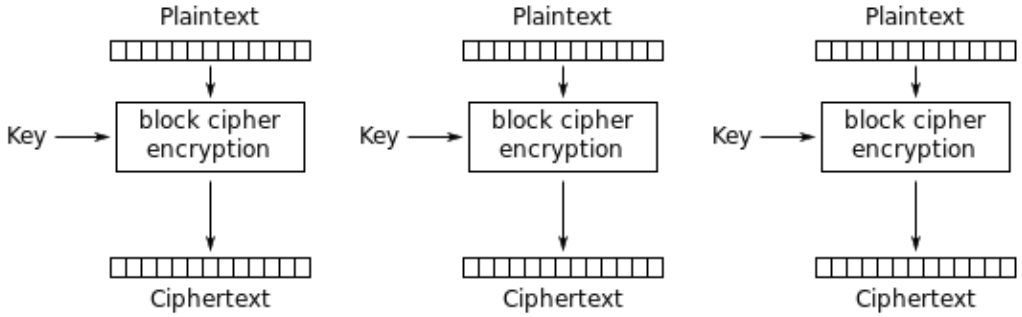


Figure 2 - Electronic Book Code (EBC) mode encryption

For the two following questions, you are asked to encrypt the following file : `./cbc/clearTextCBC`

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

Application Security – Lab Session 1
Java Security

laurent.gomez@sap.com

RSA Signature Implementation

(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/SqueletonEntity.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 keypairgenerator object of type `java.security.KeyPairgenerator` for RSA.
- Generate a keypair public/private.
- 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` ».
- Initialise 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*
- Initialise it in *VERIFY_MODE* mode with the public key
- Check the signature.

laurent.gomez@sap.com

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

3.1. RSAEncryption (1 point)

Complete method *`Entity::encrypt()`*

3.2. RSADecryption (1 point)

Complete method *`Entity::decrypt()`*.

laurent.gomez@sap.com

---

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 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 the slides from the application security lecture for further details on this secure session key exchange.