DATA PROTECTION

Departamento de Ingeniería Telemática

2022

PRACTICAL WORK: OBJECTIVES

The objectives of this practical work are the following:

1. **To study** and **understand** modern ciphers and digital signature mechanisms. As an example and with the knowledge gained, to **implement** a cryptosystem based on the combined application of a public-key cipher (RSA) and a symmetric cipher (AES).

2. **To implement** a data protection application, capable of encrypting/decrypting files that are locally stored in the user equipment. Additionally, the application will support the execution of signing operations, as well as signature verification, over these files.

GENERAL RULES

The practical work is structured in three sections each of which contains a proposal for the implementation of a set of software components.

NOTES:

- ✓ This work can be done individually or in groups of two.
- ✓ Faced with any unspecified aspect of this exercise the student should make any implementation decisions that are considered appropriate. Such decisions should be justified during the evaluation of the program performed by the teacher and the author or authors of the program.
- ✓ The implementation must work on the laboratory equipment, either in Windows or Linux operating systems (standard JDK)
- ✓ To facilitate the implementation, students are provided with additional Java material for each of the three sections. Read each section in particular for details.
- ✓ All the students should deliver the different sections in 'Aula Global' Tasks. The details of the deliveries (schedule, format) will be explained by the teacher.

SECTION 1: *symmetric encryption library*

The objective of this part is to implement a Java library that provides the necessary functions to encrypt and decrypt arbitrary length messages using the symmetric AES (Advanced Encryption Standard) algorithm in one of its modes of operation. For this, the student will implement the following library functionalities:

1. **Encryption of arbitrary length texts**. This functionality allows you to encrypt texts of any length using **AES in CBC** (Cipher Block Chaining Mode) operation.

   Since the symmetric key algorithms, based on the concept of a block cipher, allow you to encrypt blocks of plaintext of fixed size, it will be necessary to apply some padding technique to the input file to make its length be a multiple of the size of the block. In this case, **the library will implement the padding technique PKCS#5** [1].

2. **Decryption of texts of arbitrary length.** This functionality will be complementary to the previous one. It will allow you to decrypt texts of any length that have been coded with AES in CBC (Cipher Block Chaining Mode).

   After the decryption process, the implementation must eliminate the padding introduced in the encoding process.

NOTES:

- ✓ To simplify the coding of this library a Java class (named *SymmetricEncryption*) is provided as additional material, which implements a basic AES encrypting/decrypting functionality. This class has been built using the *javax.crypto* package of Java, Appendix A provides detailed information on this library as well as several examples of its use.

- ✓ To easy the implementation, the student is provided with a Java class, named *SymmetricCipher*, which defines the set of methods that must be implemented. The students must complete the implementation of this class to achieve the aforementioned functionalities.

- ✓ The coding of the mode of operation CBC must be done by the student, using only the AES library provided in the complementary material. The student must not use any existing implementations of AES/CBC.

- ✓ The detailed description of the *javax.crypto* package is available in the Java Platform API Specification (https://docs.oracle.com/javase/8/docs/api/). The student is referred to this specification to solve any questions related with the operation of this Java package.

- ✓ It is strongly recommended that the students include into the delivery an explanation of how the tests have been made. For instance, if a 'Test.java' with the Main class has been coded, it is recommendable to include it in the delivery

SECTION 2: a*symmetric encryption library*

In this section the student will implement a Java library with methods to support the encryption and decryption of arbitrary length plaintexts using the asymmetric key algorithm RSA (Rivest, Shamir, Adleman). This library will provide the following functionality:

- **Generation of 1024 bit RSA keys**. The asymmetric encryption library will include a method to generate a pair of RSA keys, a public key and its corresponding private key. Each of these keys will be 1024 bits long. As a result of the execution of this method, the following files will be created:
  - "*public.key*": this file will contain the RSA public key that has been created as a result of the exeution of the method.
  - "*private.key*": this file will contain the RSA private key corresponding to the previous public key.

  Subsequent calls to this method will delete the existing RSA keys and generate a new key pair.

- **RSA cryptographic functions**. The library will include methods to support the encryption and decryption of texts of arbitrary size, using the RSA algorithm and an RSA key pair.

NOTES:

- ✓ To simplify the implementation, the student is provided with a Java class, named *RSALibrary*, which provides a partial implementation of the methods to support the generation of RSA keys, as well as the RSA encryption and decryption operations. The source code of this class, included in the file *RSALibrary.java*", contains a description of each of the methods to be implemented.

- ✓ The students must complete the implementation of the *RSALibrary* class to achieve the aforementioned functionalities. The code that must specifically be implemented by the student is indicated in the Java file with the label "TO-DO".

- ✓ The files "*public.key*" and "*private.key*" will be created in the same folder that stores the bytecode of the Java library *RSALibrary*.

- ✓ For the implementation the student can only use the classes and methods included in the Java packages *java.security* and *javax.crypto*.

- ✓ The detailed description of the Java packages *java.security* and *javax.crypto* is available in the Java Platform API Specification (https://docs.oracle.com/javase/8/docs/api/). The student is referred to this specification to solve any questions related with the operation of these Java packages.

- ✓ It is strongly recommended that the students include into the delivery an explanation of how the tests have been made. For instance, if a 'Test.java' with the Main class has been coded, it is recommendable to include it in the delivery

SECTION 3: *application for secure storage of files*

Finally, with the support of the symmetric and asymmetric encryption provided by the libraries of previous sections, the student will **code an application** that will allow the secure storage of files in the local hard disk of the computer. The application will support the encryption and signing of files, allowing for confidentiality, authenticity, integrity and non-repudiation of the stored data.

The application will be programmed in a class with name *SimpleSec*. The execution of the *bytecode* will follow the syntax:

java ***SimpleSec*** *command [sourceFile] [destinationFile]*

The parameter *command* may take the following values:

- "***g***": the application will generate a pair of RSA keys, a public key and its corresponding private key, using the Java library programmed in Section 2.

  To protect the private key generated by the application, the user will be prompted to introduce a ***passphrase***. The text string provided by the user as the passphrase will be used as an AES key to encrypt the file "*private.key*" in the local hard disk, using AES/CBC. This way, the private key is protected, only being accessible to the legitimate user that has generated it.

- "***e***": the file indicated by *sourceFile* must be encrypted and signed, using the libraries programmed in the previous sections. Appendix B describes the specific procedures that must be implemented by the student for the encryption of files (the signature can be generated by simply using the RSA library programmed by the student in Section 2).

  As the signature generation will need the private key of the user, the user will be prompted to introduce its *passphrase*. The text string provided by the user as the *passphrase* will be utilized as an AES key to decrypt the file "*private.key*", and to retrieve the private key.

  The result of the encryption and signing processes is stored in a new file, in the location determined by *destinationFile.*

- "***d***": the file indicated by *sourceFile* must be decrypted, and the signature contained in this file must be verified, using the libraries programmed in the previous sections. Appendix B describes the specific procedures that must be implemented by the student for decryption (the signature can be verified by simply using the RSA library programmed by the student in Section 2).

  As the decryption process will need the private key of the user, the user will be prompted to introduce its *passphrase*. The text string provided by the user as the *passphrase* will be used as an AES key to decrypt the file "*private.key*", and to retrieve the private key.

  The operation should fail in case that the signature cannot be verified. In case of successful verification, the result of the decryption is stored in a new file, in the location determined by *destinationFile*.

NOTES:

- ✓ Under no circumstance the passphrase provided by the user be stored on disk (it is a password that the user should remember).

- ✓ The student must take any motivated decisions related to the specific format of an encrypted/signed file.

- ✓ It is mandatory to deliver the source code. It is strongly recommended to include a .jar into the delivery, to avoid problems with the compilation of the files

REFERENCES

[1] B. Kaliski. "*PKCS #5: Password-Based Cryptography Specification. Version 2.0*". RFC 2898. RSA Laboratories, September 2000.

[2] William Stallings, "Cryptography and Network Security: Principles and Practice", Prentice Hall Pearson Education, ISBN: 9780273793359, 2013.

[3] PGP (*Pretty Good Privacy*). http://www.pgp.com

APENDIX A
Basic cryptographic operations library

As complementary material, a Java class (*SymmetricEncryption*) is provided, with a set of basic symmetric cryptography functions that must be used for this practical work. This class has been built using the *javax.crypto* package of Java, and implements a block cipher that uses the AES algorithm with keys of 128 bits. In this appendix we describe the most important methods contained in the *SymmetricEncryption* class.

Methods

public ***SymmetricEncryption*** *(byte[] byteKey)*

This is the constructor method of the class. It allows initializing the AES block cipher to operate with a symmetric key with a length of 128 bits.

Arguments:
- *byteKey*: byte array holding the encryption/decryption key that is to be used by the AES block cipher.

Exceptions:
- *InvalidKeyException* (detailed information about this exception can be found in the Java Platform API Specification, *https://docs.oracle.com/javase/8/docs/api/*).

public byte[] ***encryptBlock****(byte[] input)*

This method implements the symmetric key encryption algorithm AES. It receives as input a single 128 bit plaintext block (16 bytes) and produces a single 128 bit encrypted block as output. Only one 128 bit block can be encrypted per function invocation.

Arguments:
- *input*: byte array holding plaintext block of 16 *bytes* to be encrypted. The *input* variable must contain exactly than 16 *bytes*.

Result:
- A byte array holding the cryptogram, in case that the AES encryption operation has been successful.

Exceptions:
- *IllegalBlockSizeException*, *BadPaddingException*. (detailed information about these exceptions can be found in the Java Platform API Specification, *https://docs.oracle.com/javase/8/docs/api/*).

public byte[] ***decryptBlock****(byte[] input)*

This method implements the decryption of a 128 bit block (16 *bytes*) encrypted with AES. It receives as input a single 128 bit cryptogram (16 bytes) and produces the corresponding 128 bit decrypted block as output. Only one 128 bit block can be decrypted per function invocation.

Arguments:

- *input*: byte array holding plaintext block of 16 *bytes* to be encrypted. The *input* variable must contain exactly than 16 *bytes*.

Result:

- A byte array holding the cryptogram, in case that the AES encryption operation has been successful.

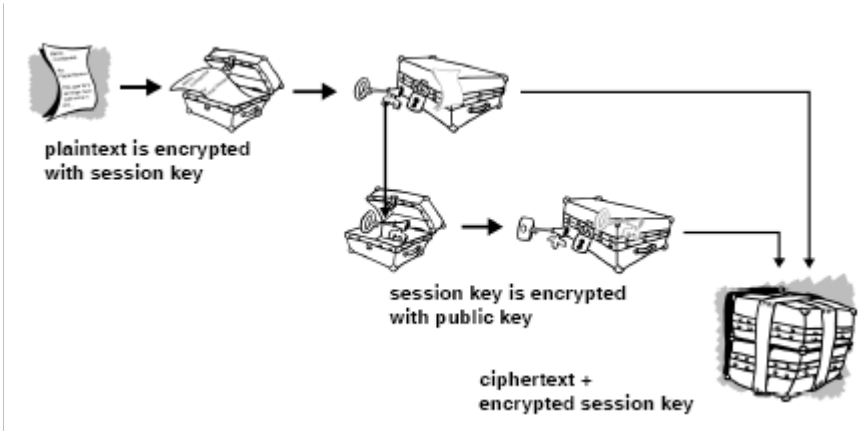
Exceptions:

- *IllegalBlockSizeException*, *BadPaddingException*. (detailed information about these exceptions can be found in the Java Platform API Specification, *https://docs.oracle.com/javase/8/docs/api/*).

APENDIX A
Summary of cryptographic operations to be implemented

Encryption operation

The encryption operation to be implemented will be similar to the PGP implementation [3], particularized with the use of RSA as the asymmetric encoding algorithm and with AES/CBC as the symmetric encoding algorithm. The encryption scheme is presented in the following figure:



Basically, the encryption scheme consists of the following steps:

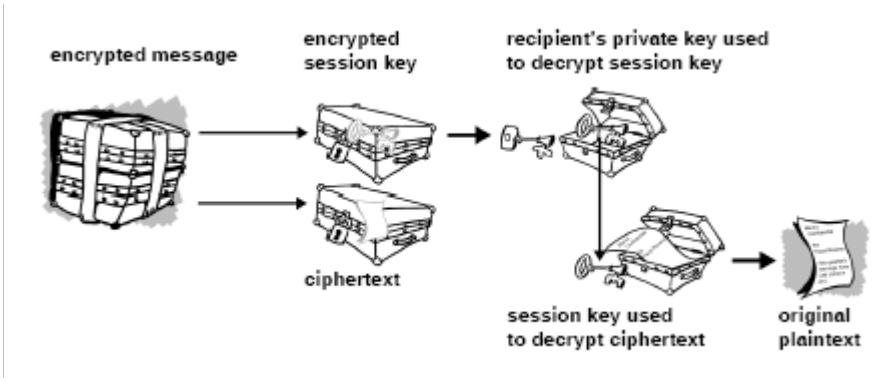
1. Encrypting the input text, using the symmetric encryption algorithm AES in CBC mode of operation. For the encryption a randomly chosen AES key will be used.

2. Encryption of the AES key used in step 1. For this the asymmetric RSA algorithm will be used with the public RSA key of the user.

3. The resultant encryption of this scheme will consist of the concatenation of the encryption obtained in step 2 and the encryption obtained in step 1 in a single output text.

Decryption operation

This operation will allow decrypting a file encrypted with the encryption operation. The decryption scheme is presented in the following picture:



Basically, the scheme consists of the following steps:

1. The input encrypted text is divided in two parts: the encrypted text corresponding to the session key and the encrypted text corresponding to the original plaintext.

2. Decryption of the session key. For this decryption the asymmetric algorithm RSA will be used with the RSA private key of the user. The application will prompt the user for the *passphrase* necessary to retrieve the private RSA key.

3. Decryption of the encrypted text, using the symmetric decryption algorithm AES in CBC mode of operation. For the decryption process the AES key obtained as the result of the step 2 will be used.