Hacettepe University
Department of Computer Engineering
BBM465 Information Security Laboratory
Experiment 2

Subject:	Asymmetric Cryptography, Hashing and Digital Signatures
Language:	Java
Due Date:	18/11/2019 - 23:59
T.A. :	Dr. A. Selman Bozkır

1 Aim and Background

In this project your aim will be development of a licensing framework by utilizing the methods of asymmetric cryptography, MD5 and digital signatures. Although the methods take place in symmetric cryptography are strong, they have some shortcomings such as key publication for both *receiver* and *sender*. This could be a hard procedure if these two counterparts live in different cities or they are geographically far away without having a secure communication. Moreover, symmetric cryptography is prone to "man in the middle" attacks which serves a potential vulnerability of changing the content of the message on the way to the receiver.

On the other hand, the asymmetric cryptography presents a different perspective in order to solve these drawbacks. First, it offers to use a publicly available *public key* for encryption of the message which will be only decrypted by using the *private key*. Note that, these two keys forms a *key pair* and, along with public key, only the receiver holds the private key for tasks of (1) decryption and (2) digital signature creation. The receiver is able to decrypt the encrypted message and it can create a digital signature of a content for further authentication and verification purposes.

2 Experiment

As is stated, in this experiment, you are expected to develop of a licensing framework by utilizing the methods of asymmetric cryptography, MD5 and digital signatures. The following lines will state the requirements of your assignment. Besides, you can view a schematic representation and workflow of the desired system in Figure 2.

- Let us assume that you are supposed to create a licensing module for a highly costed software project. According to the requirements, you must design a module which will be located into the software itself (the *Client*) and a server system (the *licence manager*). In practice, this scheme is implemented via a network system such (e.g. client and web server). Further, *clients* do communicate with a server system through web services. In this assignment, instead, you will do a real world simulation of this scheme by creating 2 main classes in Java namely "Client" and "LicenseManager".

- According to the requirements, the *client* module must first check the existence of the "license.txt" in the application folder. The license.txt file involves a digital signature signed by the *licence manager*. If the license.txt exists, then your client module must verify it by using the public key which is given to you at startup. Note that, the client holds only the private key ("private.key" as the file name) whereas license manager holds both the "private.key" and "public.key" at local.

- If the "license.txt" does not exists, then this indicates that, the software has never been licensed before. In order to license the application your first duty is to collect the following identities:

  1. the username (string)
  2. the serial number (string having format of ####-####-####)
  3. MAC address of the Ethernet device of the host system (string)
  4. Disk serial number (string)
  5. Motherboard ID (string)

  For username and serial number you can use a static text. In other words, you experiment will not ask the user to input theses data by hand. Here, they are only given to provide a perspective. For the other device specific data, you can use third party codes. Nonetheless, you can not use JNI or JNA based *native* services or packages. More precisely, you should use core Java library based solutions for this step. Once you collect these data, you need to concatenate them by using $ as a separator character. An example is given below for this plain text ("user-serialid-hw spefic info" tuple) content.

  Example of a "user-serialid-hw spefic info" tuple:
  Selman$0H6U-23BJ-YR84$0C-54-15-5B-0A-FE-633475686Standard

- After forming the hardware and user specific unique plain text, you must encrypt this content via the public key (the RSA algorithm will be used as the basis). Following this encryption you need to obtain an encrypted version of this content which has been exemplified below:

  Example of an encrypted message:
  H9 b??Gl????j????W???̂??=2?)4i????=?—?????j;Z???Y???!cI4?

- Upon having encrypted, the encoded message must then be sent to *licence manager* for further processes. Once the the server (the license manager) has received the encrypted package it must decrypted the content in order to reveal the user and serial number info along with hardware specific ids.

- Licence Manager can then do several operations such as storing these fields into a database. However, for simplicity we skip these stages. Rather the license manager must create a hash of the revealed plain text content. For this purpose, you must use MD5 (i.e. Message Digest - an irreversible message digest mechanism) to produce the hash. An example hash content has been presented below:

  Example of a MD5 hash:
  8933b3eac77b03fd8274fc41860ffcd0

- Following the hash production, you must sign the hash content via the private key. For this procedure, you can utilize "Signature" class (provided by the Java Security API) through "SHA256WithRSA" scheme. This will create the digital signature of the *hash* to be delivered to the *Client*.

- The response (i.e. digital signature) of the license manager must be verified in the *Client* module with the hash provided by *Client*. To do so, you must do the same procedure (i.e. hashing the plain user-serial-hw specific data) in the client side as well. The verification scheme works only in this way. It is important to keep in mind that, the verification must be carried out by using the public key.

- If the verification of the digital signature signed by the Licence Manager, then you should prompt the user for the success of the operation and store the digital signature into the "license.txt" . With this step, whole licensing stage finishes. You must prompt the user with the message of

  "Succeed. The license file content is secured and signed by the server."

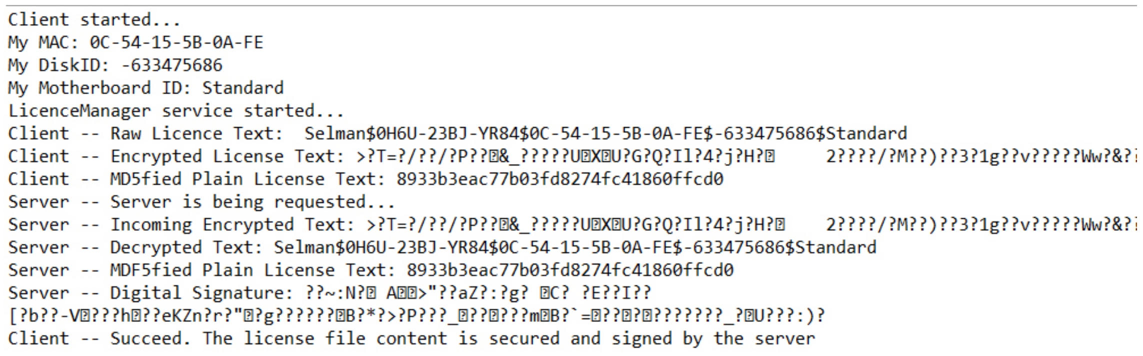
- As you may question, will this whole process be executed every time? The simple answer is NO. Instead, the only thing you must do is to check the existence of the "license.txt" in the project folder at start-up and verify its content (i.e. build the plain "user-serialid-hw specific info" tuple at client side and hash it via MD5) along with the digital signature. If the result of the verification is True then you can inform the user by prompting

"Succeed. The license is correct."

If the content of the "license.txt" is destructed by an attacker, the verification fails. In this case, you should warn the user by prompting

"The license file has been broken!!"

and re-execute the licensing process again to obtain a valid digital signature. The output of a working instance of the desired system has been depicted in the figure below:



```
Client started...
My MAC: 0C-54-15-5B-0A-FE
My DiskID: -633475686
My Motherboard ID: Standard
LicenceManager service started...
Client -- Raw Licence Text:  Selman$0H6U-23BJ-YR84$0C-54-15-5B-0A-FE$-633475686$Standard
Client -- Encrypted License Text: >?T=?/??/?P??▯&_?????U▯X▯U?G?Q?Il?4?j?H?▯     2????/?M??)??3?1g??v?????Ww?&?
Client -- MD5fied Plain License Text: 8933b3eac77b03fd8274fc41860ffcd0
Server -- Server is being requested...
Server -- Incoming Encrypted Text: >?T=?/??/?P??▯&_?????U▯X▯U?G?Q?Il?4?j?H?▯    2????/?M??)??3?1g??v?????Ww?&?
Server -- Decrypted Text: Selman$0H6U-23BJ-YR84$0C-54-15-5B-0A-FE$-633475686$Standard
Server -- MDF5fied Plain License Text: 8933b3eac77b03fd8274fc41860ffcd0
Server -- Digital Signature: ??~:N?▯ A▯▯>"??aZ?:?g? ▯C? ?E??I??
[?b??-V▯???h▯??eKZn?r?"▯?g??????▯B?*?>?P???_▯??▯???m▯B?`=▯??▯?▯???????_?▯U???:)?
Client -- Succeed. The license file content is secured and signed by the server
```

Figure 1: The output of a working instance in case of licensing process

3 Important Notes

In this chapter, for the sake of clarification, some important notes about your implementation have been described.

- You will be provided with the *public.key* and *private.key* via Piazza system.
- Your aim will be to design 1 project including these functionalities.
- Your code environment will be limited with 2 files named "main.java" and "licensemanager.java". Here the main.java will be the main "Client" holding the public key owned by the "License Manager". On the other hand, "Licence Manager" can access to both of these files.
- For simplicity, you can store the user name and serial id into a file (i.e. "user_serial.txt" for retrieving the them quickly.

- The Figure 1 exemplifies the system messages during the events. As you can see, the license manager known as *server* side events are listed as "Server –". Similarly, the *Client* side events have been shown as "Client – ". You should pay attention to the this format and apply the same scheme for your own implementation.
- You can only use Java's own byte-to-string and string-to-byte helper functions. In other words, you are prohibited to employ third part JAR files and helper classes.

4 Remarks

1. In Java, you must use standard crypto API.
2. You should prepare a report involving your approach and details of the implementation you have coded. You must write down the names and ids of your teammates in the report in order to be evaluated correctly.
3. You can ask questions about the experiment via Piazza group (`piazza.com/hacettepe.edu.tr/fall2019/bbm465`).
4. Late submission will not be accepted!
5. T.A. as himself has right to partially change this document. However, the modifications will be announced in the Piazza system. In case, it is your obligation to check the Piazza course page periodically.
6. For Java, you must compile and test your code on Eclipse Platform for Windows before submission.
7. You are going to submit your experiment to online submission system: `www.submit.cs.hacettepe.edu.tr`

   The submission format is given below:
   <Group id>.zip
   −[main.java]
   -[*licensemanager.java*]
   −[user_serial.txt]
   -[*report.pdf*]

5 Policy

All work on assignments must be done with your own group unless stated otherwise.You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That

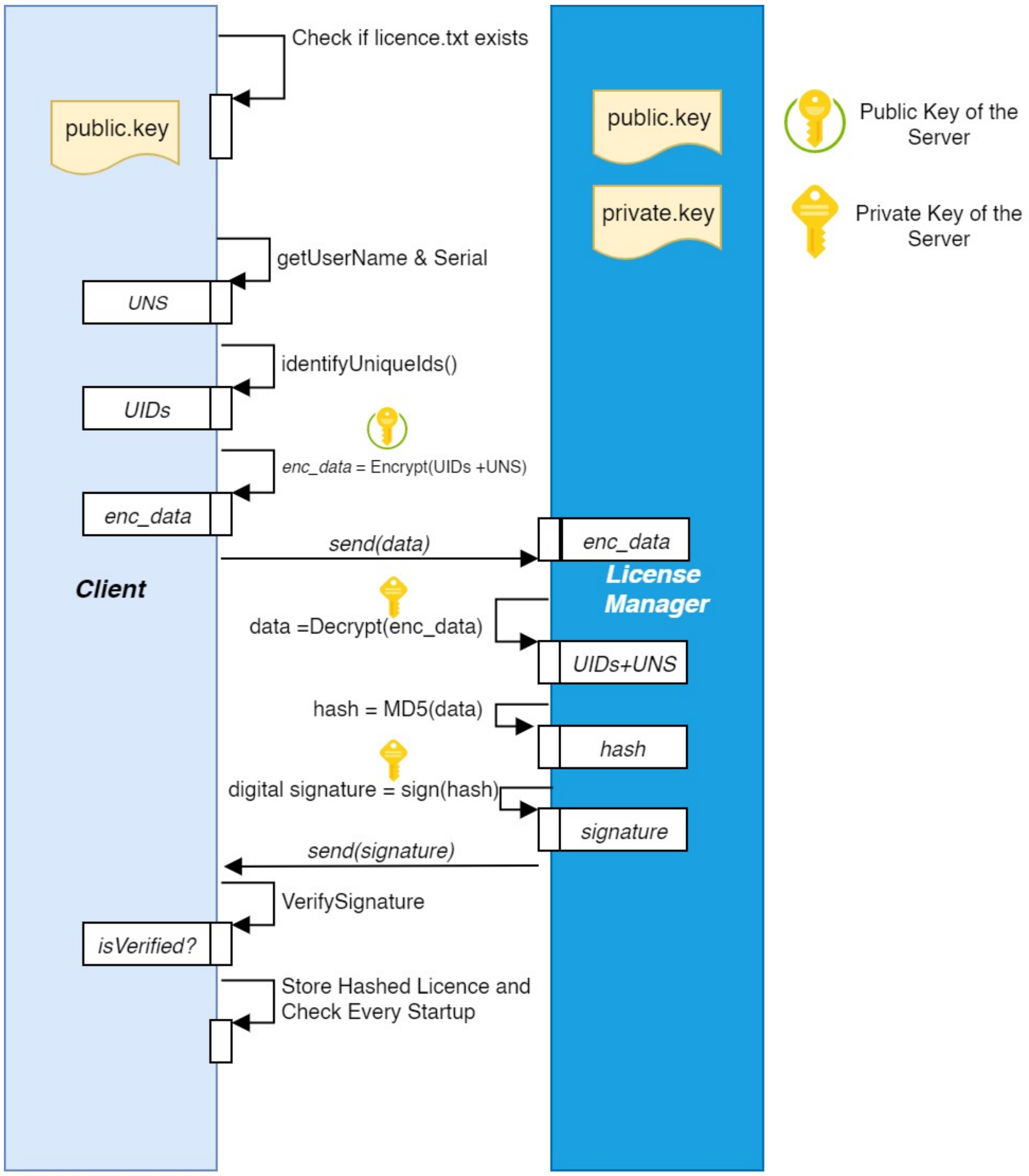


Figure 2: The workflow of the system

is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudo code) will not be tolerated. In short, turning in someone else's work (from internet), in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.