National University of Singapore
School of Computing
CS2105: Introduction to Computer Networks
Semester 1, 2015/2016

Assignment 3
Cryptographic Messages

Release date: 23 October 2015
Due: 15 November 2015, 23:59

Overview

In this assignment, you will get a taste of using the Java APIs for cryptography. You will take the role of Alice, write the code to receive an encrypted message from Bob.

Plagiarism Warning

You are free to discuss this assignment with your friends. But, ultimately, you should write your own code. If you wish to use code obtained online or some other sources, you should put a comment to credit the portion of code that you did not write. This will allow us to judge the marks which you should be awarded. We employ a zero-tolerance policy against plagiarism. If a suspicious case is found, student would be asked to explain his/her code to the evaluator in person. Confirmed breach may result in zero mark for this assignment and further disciplinary action from the school.

Description

Bob wants to send a confidential document to Alice.

Step 1. Alice generates an AES session key for the session and encrypts it using Bob's public key which she has in her possession. Alice then sends this encrypted key to Bob.

Bob then uses his private key to decrypt and obtain the session key.

Step 2. The confidential document that Bob has is a text file of exactly 10 lines. For each line, Bob encrypts it with the AES session key, and sends it to Alice as a `SealedObject`.

Step 3. Alice then decrypts each SealedObject and saves each line to a plaintext file.

The program `Bob.java` is written and given to you for your reference. Your task is to complete the skeleton program `Alice.java` to fulfil the task mentioned above.

`Bob.java`

This program should not be modified or changed. We will run the original version in our testing. To run the program, type "`java Bob <port>`" where `port` is the TCP port on which Alice will connect to.

On startup, Bob will read his RSA private key in the file `bob.pri` stored in the current directory. Upon receiving a TCP connection from Alice, Bob will first receive the encrypted session key and decrypt it with his private key. He will then read and encrypt the file `docs.txt` line by line with the received session key and send them to Alice.

You should read `Bob.java` carefully as you are supposed to write the counterparts in `Alice.java`.

`Alice.java`

To run the program, type "`java Alice <IP address> <port>`" where `IP address` and `port` is the IP address and port where Bob is listening to.

On startup, Alice will read Bob's RSA public key from the file `bob.pub` which is stored in the current directory. The contents of the file was serialized and written using Java `ObjectOutputStream`. Thus, it is to be read using Java's `ObjectInputStream`.

Alice will then initiate a TCP connection to Bob and when connected, generate an AES key, encrypt it with Bob's public key and send it to Bob. She will then receive 10 encrypted messages from Bob, and decrypt and save them the file `msgs.txt` in the current directory.

Advanced Criteria

The above description describes the basic criteria which if fulfilled, will earn you 9 marks in total. For stronger students who wants more challenge, the advanced criteria will get you the full 13+1 marks for the assignment.

In the previous scenario, Alice is able to securely send the generated session key to Bob as she has Bob's public key. In this scenario, Bryan wishes to send a message to Amy, but Amy does not have Bryan's public key, nor does she have a secure channel to Bryan. Bryan can always send Amy his public key but Amy is worried that Trudy might intercept Bryan's transmission and substitute her own key instead. Amy needs a way to verify that Bryan's key really belong's to him.

Fortunately, Amy trusts a third-party called Berisign and has Berisign's public key. Berisign also has a secure means of communicating with Bryan. Berisign, in this case acts as a Certificate Authority and can sign public keys. To keep things simple, we will use our own simplified protocol to do perform this process. The actual process in real life involves more complicated objects which we will not touch in this assignment. The protocol works as follows:

Step 1: Bryan has got Berisign to sign the MD5 digest of his public key and his for verification purpose. The MD5 digest was created by first updating with an ASCII string "bryan" followed by the MD5 sum of his public key as follows:

```
// md5 is a MessageDigest using MD5 algorithm
md5.update(name); // name the is US-ASCII byte encoding of the string "bryan"
md5.update(public_key);
byte[] digest = md5.digest();
```

Berisign then encrypts the MD5 digest with its private key to a `SealedObject`, which it passes to Bryan to distribute.

Step 2: When Amy connects to Bryan, Bryan sends Amy his RSA public key, followed by the `ByteArray` containing the encrypted signature which Amy can decrypt using Berisign's public key. She can then perform the same process to obtain the MD5 sum and check if it matches the one contained in the `ByteArray`. Since only Berisign has the private key needed to encrypt the MD5 sum, Amy is confident that the public key belongs to "bryan".

Step 3: Having obtained and verified Bryan's public key, Amy and Bryan then proceeds on to generate and exchange the session key and the message as described in the basic criteria previously.

`Bryan.java`

The program `Bryan.java` is given to you. It is basically modified from `Bob.java` in that Bryan sends his public key and signature at the start of the connection.

`Amy.java`

You are to write `Amy.java` which is run the same way as `Alice.java`. You may modify `Alice.java` to accommodate the new requirements.

Java Cryptography Classes

The `Cipher` class in the Java API forms the core of the Java Cryptographic Extension (JCE) framework and provides the functionally for encryption and decryption. `Cipher` object needs to first be initialized to `ENCRYPT_MODE` or `DECRYPT_MODE` before performing the respective operations. The encryption algorithms such as RSA, AES and DES are supported. There is no need to study the feedback mode and padding scheme of these algorithms.

In our secure transmissions, we will encapsulate and send the messages as Java `SealedObjects`. `SealedObjects` are containers like a locked box which can be encrypted or decrypted with the help from a `Cipher` object. The reason we use `SealedObjects` is that they implement the `java.io.Serializable` and can be easily sent over TCP using ObjectInput/OutputStreams. Otherwise, you would to transmit byte arrays over TCP, for example, using the `doFinal` method of the `Cipher` class to return the encrypted message as a byte array.

The `KeyGenerator` class is used to generate an AES `SecretKey` object. Alice will also encapsulate the AES key as a `SealedObject` when transmitting it to Bob. However, sealing an AES key in a `SealedObject` using RSA presents a problem—the RSA algorithm imposes a size restriction (typically 117 bytes) on the object being encrypted. An AES `SecretKey` object is too large to fit. The solution is to instead seal the encoded form of an AES `Key` object using the `getEncoded` method. Bob will then reconstruct the AES `Key` object from the received encoded form.

Please read the relevant Java API documentation for more information on these Java classes. You are also recommended to study `Bob.java` as a reference. There is no need to use any additional Java cryptography classes that is not imported in the given skeleton for the basic criteria.

Python Classes

The `PyCrypto` module is used to support cryptography in Python. You can refer to the API documentation and use the code snippets from here https://www.dlitz.net/software/pycrypto/api/current/.

We have also provided a AESCipher class for easy creation and encryption/decryption using AES symmetric cryptography. This is because AES requires the length of the data to be encrypted to be a multiple of the block size. The class provided does the padding and unpadding for you. All you need to create a AES key is a 32-byte password which can be randomly generated using the `Crypto.Random` class.

The RSA keys are exported to text files using the `exportkey()` function and can be imported using `RSA.importkey()`. Again, the raw RSA key requires some work for the actual encrypting and decrypting so we will use the `PCKS1_OAEP` scheme included in the `PyCrypto` module.

There is no `SealedObject` equivalent in Python (at least not by default). We will instead use `pickle` to serialize the encoded objects for sending through the socket. You can again find some code snippets on the web which you can freely use.

The PyCrpyto library also comes with modules for verifying RSA signatures. For the advanced criteria, the signature of Bryan's public key is signed by Berisign's private key using the `PKCS1_PSS` signature scheme. Again, you can find the documentation in the API. The steps which produced the signature is as follows:

```
# md5 is a Hash object using MD5 algorithm
md5.update("bryan")
md5.update(pubkey.exportKey())
signer = PKCS1_PSS.new(prikey)
signature = signer.sign(md5)
```

You can then verify if the signature matches the digest using the `verify` function.

Please include the following line to set the socket option in your code:
`sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)` to ensure that the port will be released immediately after the socket closes.

Also, note that the `PyCrypto` module is not installed on `sunfire`, so we will test it on another system. We will use Python 2.7.3 so make sure your code is compatible with it.

Python Files

Similarly to the Java implementation, `bob.py` and `bryan.py` is correct and given to you. You should not modify or change the behaviour of the programs as we will run the original version for testing. We advise you study these files and use it as reference as you will be writing the counterpart.

A skeleton code is given in `alice.py`. To avoid confusion with the Java implementation, the key files are named differently. But the text files `docs.txt` and `msgs.txt` will remain the same.

Submission

You will submit your program on IVLE. Zip your files into a single zip file and name it "`a3-<student number>.zip`" and upload to the Assignment 3 folder. For example, if your student number is A0123456X, then you should name your file "`a3-a0123456x.zip`".

It is your responsibility to ensure that your file is successfully uploaded into IVLE before the dateline. You should refresh the IVLE workbin and confirm that your file there.

The dateline for submission is at 15 November 2015, 23:59 hrs.

Submitting in Java

Your zip file should include the java files `Alice.java` (and Amy.java). You need not include Bob or Bryan as we will supply the original version when testing. You may include other necessary java files in the zip file.

We will unzip the contents of your zip file and compile your submission using `javac *.java` and run each program as described above.

Submitting in Python

You may write your program in Python. Your zip file should contain the files `alice.py` (and `amy.py`). You do not need to include `AESCihper.py`, `bob.py` or `bryan.py` as we will supply the original version when testing.

C++

Because there is no standard C++ cryptographic libraries, we will not be supporting C++ for this assignment. It is also not easy to write cryptographic code from scratch either. We advise you try Python as it is very simple and a straightforward language to use.

Details and Grading

The score for this assignment is 13 marks. If you obtain above 13 marks, half of the extra marks can be passed-on to other assignments.

Late submissions will be subject to the following penalties:

- 0 - 24 hrs late: Marks capped at 12.
- 24 - 48 hrs late: Marks capped at 10.
- beyond 48 hrs: Marks capped at 8.

The absolute deadline for all submissions will be on 20 November 23:59. As this is very close to the end of the semester, there will be no avenue of redress after this dateline.

Basic criteria:

- `Alice.java` can be compiled on `sunfire`, or `alice.py` runs with no errors.**(1 mark)**
- `Alice.java` successfully runs using the specified Java command, or `alice.py` successfully connects to Bob. **(2 marks)**
- `Alice.java` or `alice.py` correctly decrypts the messages and saves them to the specified file `msgs.txt`. We do not care what your program prints to the screen, as we will only check that the generated file is identical to what Bob has. **(6 marks)**

Advanced criteria:

- `Amy.java` or `amy.py` correctly decrypts the messages from Bryan and saves them to the specified file `msgs.txt`. We do not care what your program prints to the screen, as we will only check that the generated file is identical to what Bryan has. **(3 marks)**
- `Amy.java` or `amy.py` correctly detects that the MD5 sum of the signature does not match the received public key and outputs the string `"Error:MD5 signature does not match"` before terminating. We will only check the last line output by your program so you can still output other comments for tracing. **(3 marks)**