National University of Singapore
School of Computing

CS2105 **Assignment 3** Semester 1 AY19/20

Introduction and Submission Deadline

In this assignment, you will implement two programs that use Java API/Python library methods for cryptography. This programming assignment **is worth 6 points.**

The deadline of submission is **13 November 2019 (Wednesday), 11:59pm** sharp. Do not leave your submission to the last minute in case of unforeseeable situation. You may submit many times and only the last submission will be graded.

2 points penalty will be imposed on late submissions (i.e. submissions or re-submissions made after the deadline). **No submission will be accepted after 20 November 2019 and 0 point will be awarded.**

Grading

We will test and grade Java programs on the `sunfire` server, and Python3 programs on the **`xcna0`** server, as the PyCrypto module is **not** installed on `sunfire`. Please make sure that your programs run properly on `sunfire` or `xcna0`. To access `xcna0`, please view the file "Accessing_xcna0.pdf" in the package. Moreover, you are allowed to use libraries installed in public folders of `sunfire` or `xcna0` (e.g. `/usr/lib`) only.

We accept submission of Python 3 or Java programs, and we recommend that you use **Python 3** for your assignments. Programming languages other than Python 3 and Java are not accepted. For Python 3, we use the python3 program installed in folder `/usr/bin` on `xcna0` for grading. If you use Java, we will compile and run your program using the default compiler on `sunfire` (`java 9.0.4` installed in `/usr/local/java/jdk/bin`). The grading script assumes the use of the default file extension name (.py, .java). Therefore, please ensure your files have the correct extension names. For a Java program, the class name should be consistent with the source file name, and please implement the static main() method so that the program can be executed as a standalone process after compilation. We will **deduct 1 mark** for every type of failure to follow instructions (e.g. wrong program name).

We will grade your program based on its correctness only. A grading script will be used to test your program and no manual grading will be provided.

Program Submission

For individual submission, please submit a zip file that contains source programs and submit it to the `Assignment_3_student_submission` folder of LumiNUS Files. Please name your zip file as **<Student number>.zip**, where **<Student number>** refers to your matric number that starts with letter A. Note that the first and last letters of your student number should be capital letters. One example file name would be **A0112345X.zip**. Your zip file should only contain two source programs, **Alice.py** and **Amy.py**. If you use Java, replace ".py" with ".java" respectively. Do not put your programs under any subfolders (or package paths). If you are not attempting the bonus task, there is no need to submit an **Amy.py** file.

We will **deduct 1 mark** for every type of failure to follow instructions (e.g. wrong program name, wrong zip file name, folder structure).

Grading Rubric

The marks for this assignment are as follows:

- **[2 points]** Programs is free from syntax errors and can be successfully executed (or compiled, for Java). Program execution follows the specified commands exactly. Source files are correctly named and there are no additional subfolders inside the zip file.
- **[4 points]** Alice can successfully decrypt and save the messages (in `msgs.txt`) that are received from Bob.
- **[1 point (bonus)]** Amy can successfully decrypt and save the messages (in `msgs.txt`) that are received from Bryan.

Note that the maximum number of marks for assignments is capped at **25**. As such, if completing the bonus task causes you to get more than 25 marks total for the assignments, your final marks for the assignments component will still be 25.

The messages received by Alice/Amy respectively must be **identical** to the ones sent by Bob/Bryan respectively, to claim a successful transmission. The total size of input messages to Bob/Bryan is guaranteed to be no larger than **2,000** bytes (including newline characters). During grading, Alice/Amy processes will be forced to terminate **15 seconds** after Alice/Amy is started. The grading script will then compare the saved message from Alice/Amy against the input to Bob/Bryan.

Question & Answer

If you have any doubts on this assignment, please post your questions on LumiNUS forum or consult the teaching team. We will not debug programs for you. However, we may help to clarify misconceptions or give necessary directions if required.

Plagiarism Warning

You are free to discuss this assignment with your friends. However, ultimately, you should write your own code. We employ zero-tolerance policy against plagiarism. If a suspicious case is found, we will award zero points and may take further disciplinary action.

Do not post your solution in any public domain on the Internet or share it with friends, even after this semester.

Basic Task: Alice/Bob

There are two programs involved in this task, **`Alice`** and **`Bob`**. **`Bob`** is written and given to you for your reference. Your task is to complete the Alice program to fulfil the conditions given below.

Description

Bob wants to send a message 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 pickled object (for Python) or `SealedObject` (for Java).

Step 3. Alice then unpickles each pickled object for Python, or decrypts each `SealedObject` for Java, and saves each line to a plaintext file `msgs.txt`.

`Alice` Program

To run **`Alice`** on **`xcna0`**, type the following command (suppose **python3** alias is set up):

```
python3 Alice.py <bobPort>
```

`<bobPort>` is the port number **`Bob`** is listening to. An example command line would be,

```
python3 Alice.py 9000
```

On startup, Alice should read Bob's RSA public key from the file `bob.pub` or `bob-python.pub` which is stored in the current directory. The contents of the file was written using `exportKey()` for Python, and `ObjectOutputStream` for Java, and should be read using `importKey()` for Python, and `ObjectInputStream` for Java.

Alice will then initiate a TCP connection to Bob and when connected, generate an AES session 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 in the file `msgs.txt`** in the current directory.

`Bob` Program

This program should not be modified or changed. We will run the original version in our testing. To run the program, type the following command:

```
python3 Bob.py <bobPort>
```

For example,

```
python3 Bob.py 9000
```

With the above command, Bob listens to port 9000 of localhost.

On startup, Bob will read his RSA private key in the file `bob.pri` or `bob-python.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.py` carefully as you are supposed to write the counterparts in `Alice.py`.

Bonus Task: Amy/Bryan

The above description describes the basic conditions which if fulfilled, will earn you the full 6 marks in total. For stronger students who want more of a challenge, the advanced criteria for the bonus task will get you an additional **1** mark 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, not 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 belongs 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 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 name for verification purposes. The MD5 digest was created by first updating with an ASCII string "bryan" followed by the MD5 sum of his public key (code snippets in the Python/Java classes section below).

Berisign then encrypts the MD5 digest with its private key and saves it to a pickled object (for Python), or `SealedObject` (for Java), which it passes to Bryan to distribute.

Step 2. When Amy connects to Bryan, Bryan sends Amy his RSA public key, followed by the encrypted signature which Amy can decrypt using Berisign's public key. She can then perform the same process to compute the MD5 sum and check if it matches the one sent by Bryan. 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 Task 1 previously.

Amy Program

You are to write `Amy.py` which is run the same way as `Alice.py`. You may wish to make a copy from `Alice.py`, and modify your copy to accommodate the new requirements.

Bryan Program

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

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 an `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 an 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 imported using `RSA.importKey()`. Again, the raw RSA key requires some work for the actual encrypting and decrypting so we will use the `PKCS1_OAEP` scheme included in the PyCrypto module.

We use pickle to serialize the encoded objects for sending through the socket. You can find some code snippets on the web which you can freely use.

The PyCrypto library also comes with modules for verifying RSA signatures. For the bonus task, the signature of Brian'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".encode())
md5.update(pubKey.exportKey())
signer = PKCS1_PSS.new(priKey)
signature = signer.sign(md5)
```

Also, note that the PyCrypto module is not installed on `sunfire`, so we will test it on `xcna0` instead. We will use Python 3.6 so make sure your code is compatible with it.

Java Cryptography Classes

The `Cipher` class in the Java API forms the core of the Java Cryptographic Extension (JCE) framework and provides the functionality for encryption and decryption. A `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 `java.io.Serializable` and can be easily sent over TCP using ObjectInput/OutputStreams. Otherwise, you would have 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.

For the bonus task, Berisign generates the MD5 digest of Bryan's name and 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();
```

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 are not imported in the given skeleton for the basic task.

Testing your programs

To test your program, please use your SoC UNIX ID and password to log on to `sunfire` or `xcna0`. To make the `python3` alias permanent, and thus avoiding typing the `alias` command every time you log in, you can run the following command once to store the shortcut into the configuration file for `xcna0`:

```
echo alias python3=/usr/bin/python3 >> ~/.bash_profile
```

If you test your server manually, please select a port number greater than 1024, because the OS usually restricts usage of ports lower than 1024. If you get a `BindException: Address already in use` (or similar errors for other languages), please try a different port number.

We also release a comprehensive set of grading scripts to you. There are no hidden test cases during grading. However, passing all the test cases does not guarantee that you will get full marks. We will detect fraudulent cases such as hard-coding answers.

To use the grading script, please upload your programs along with the test scripts found in the corresponding directories to `sunfire` or `xcna0`. Make sure that your programs and the test script are in the same folder. Then, you can use the following command to run the test script:

```
bash palice.sh
```

You may replace `palice.sh` with the associated test script name (`pamy.sh`, `jalice.sh`, `jamy.sh`).

To stop a test, press ctrl-c. If pressing the key combination once does not work, hold the keys until the script exits.