Computer Engineering Department.
Faculty of Engineering.
Cairo University.

Consultation and Computer Systems Security

Project Report

Project 3

Submitted by:

Ammar Alsayed Ibrahim Sec: 2 BN: 5

Project Simulation Sequence

1) Simulate the certificate authority (CA) with a directory that saving all the public keys for all users, the certificate authority public key is sended to Alice and Bob (simulate it with a file has the CA public key)
2) Alice has a message to send to Bob (simulated using stdin input or test cases messages ⇒ see run modes section)
3) Alice gets Bob's public key from the CA, and verifies that the certificate comes from the CA by the authority's public key.
4) Alice encrypt the message using RSA algorithm
5) Alice sign the message using elgamal digital signature
6) Alice send the message and the signature to Bob (simulated using files, see simio module)
7) Bob receives the message from Alice
8) Bob decrypts the message using RSA algorithm
9) Bob gets Alice's public key from the certificate authority, and verifies that the certificate comes from the CA by the authority's public key.
10) Bob verifies that the received message is from Alice using elgamal digital signature

Project modules

All modules are well-written and can be customized. You can refer to the code itself.

rsa.py

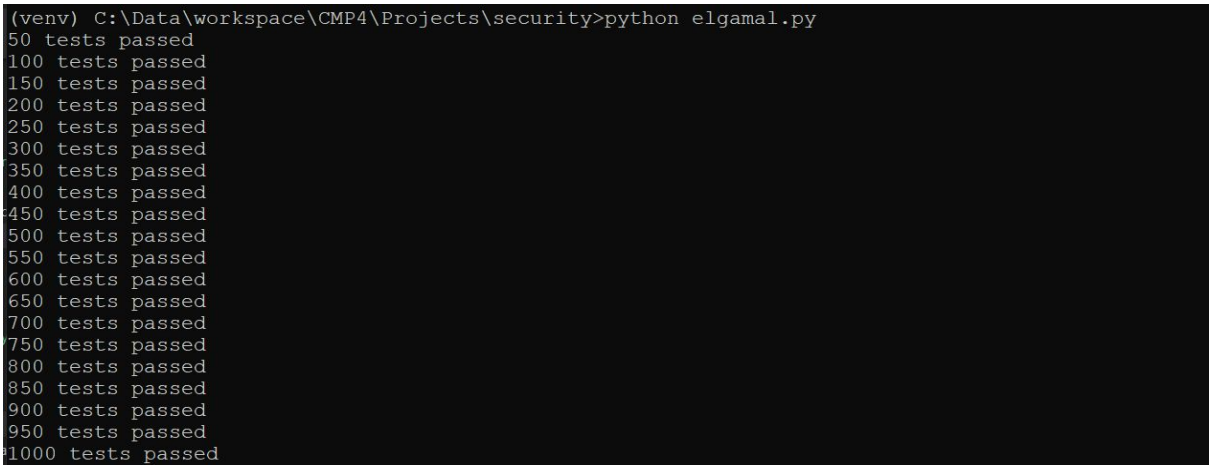
This module provides the rsa interface for other modules. You can generate asym rsa keys, encrypt, decrypt, save rsa keys, load rsa keys and make the rsa signing and verification(this is only used in CA signature, and isn't used in signature by the sender). This uses the cryptography library implementation of rsa.
This Module has unit test that can be run using **“python rsa.py”**

```
(venv) C:\Data\workspace\CMP4\Projects\security>python rsa.py
generating...
saving...
loading...
comparing bytes...
encrypting and decrypting...
Testing signing and verification..
tests passed
```

elgamal.py

This is a plain implementation of elgamal digital signature algorithm. It can

- Generate the shared parameters by default.
- Generate a user asym key using elgamal specs.
- Signing a message returning the (r, s) components of the signature
- Verifying a signature of a message
- Saving and loading the shared parameters of it
- Also it contains a unit test of it, that test upon many random keys that the signing and verification will return True if everything is ok, and test that will return False if there is a tiny change happen, you can run it by "**python elgamal.py**"



ca.py

This simulates the certificate authority. It is implemented using a singleton class that has these functionalities:

- Generate dummy certificates of readable fake names (using faker lib)
- Add certificate. This simulates adding a new certificate including verifying that the Id is belonging to the caller. A new certificate is saved to the disk.
- Get a certificate of given id. This simulates contacting the CA for a certain domain. The certificate is then returned having the desired public key and signed with certificate itself (using its private key)

simio.py

This simulates the communication between sender and receiver. This is done using files.

- Has a function that sends to a given domain. Simulated by writing the message into a new file under a directory named with that domain name
- Has a function that listens for every new message of a given domain. Simulated by listening for new files changes under that domain.

utils and others

The project has simulation_config.py that configures the paths of many files required for the simulation (e.g CA public key path, elgamal shared parameters path, …).
Utils.py also has useful common functions used by many modules.

sender.py

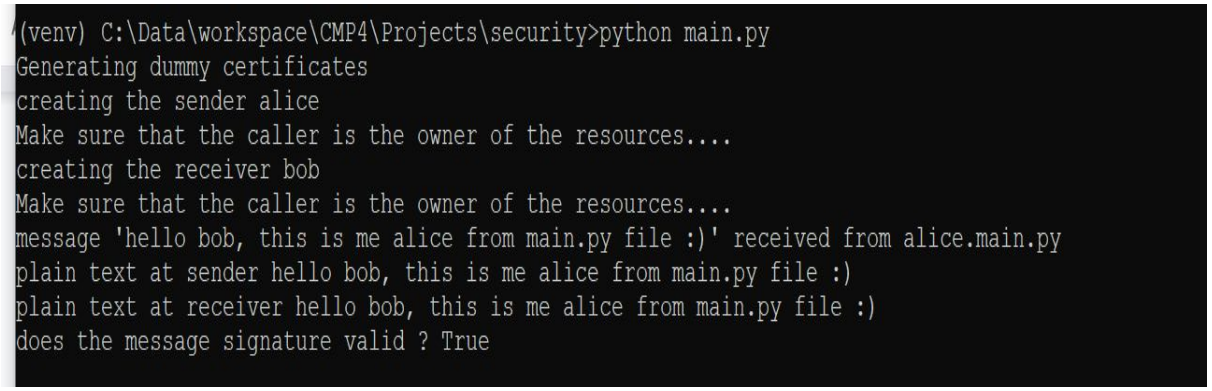
This is an isolation of a sender behaviour. If you run it specifically will make a sender process (see run modes section)

receiver.py

This is an isolation of a receiver behaviour. If you run it specifically will make a receiver process (see run modes section)

main.py

This is the project recipe. It does the steps stated in the project document. Using previous high level modules. You can consider it a silent test for the project without processes communication overhead.
Typically, it sends a message from alice to bob and make sure that the message at the receiver is the same as sended. And the signature of the message is verified correctly.
You can run it by **"python main.py"**



extra.py

This is a more tests which considers two attacks scenarios and make sure that the receiver handles them correctly

- The cipher transferred message is changed. The receiver should throw value error and can't decrypt the message
- An attacker changed the message content by making a new evil message, decrypting it using bob public key then sending it as the actual message. The receiver acts correctly and will decrypt it founding the evil message **BUT** the signature is not verified and warning message is printed

You can run it by **“python extra.py”**

General Notes

CA, Sender and Receiver classes all generate a key at their initializing of correct type. And they never save the private key in a file. Instead the private key is persisted in the object itself. Sender and Receiver classes contacts the CA to add their certificate

- CA generates an rsa key persisting its private key in the object itself and writes the public key in the file so that others can read. This key is used in signing the certificate before the CA returns it (so we make sure that the certificate is from the CA).
- Sender generates elgamal asym key. The private key is used to sign the message. The public key is read from the receiver to verify the signature.
- Receiver generates rsa asym key. The private key is used to decrypt the message. The public key is used by the senders to encrypt the message sent to this receiver.

Test cases and output

The format is under readme in output and testcases directories.

Run modes

You can run silent tests using main.py and extra.py files.

If you want to run in interactive mode and also to run the test cases, you have to follow these instructions

1) You have to run the receiver first before the sender using **“python receiver.py”**
   This will make the rid the default rec id, you can add the “--rid=yourId” option if you want.
2) Run the sender simulation using **“python sender.py testcases”**
   This will make the sid, rid to defaults, you can customize then by adding “--rid=receiverId --sid=senderId” options
   Also you can add “--nodummy” so that no dummy data created in the run

This will run the project into two processes. Please note that

- To end the sender process press enter without any message.
- To end the receiver process, press Ctrl-C, this quit it without errors.
- Don’t restart the sender process without restarting the receiver process also. And always run the receiver first.
- If you don’t want to run the test cases replace “testcases” in the command above with “run”

receiver

```
(venv) C:\Data\workspace\CMP4\Projects\security>python receiver.py --rid="rec report id"
Make sure that the caller is the owner of the resources....
message 'This is my message
The message can be multi-line
thanks all :)' received from sender report id
output saved at ./output\1_rec.txt
message 'hello, this is an interactive message from stdin' received from sender report id
output saved at ./output\2_rec.txt
```

sender

```
(venv) C:\Data\workspace\CMP4\Projects\security>python sender.py testcases
--sid="sender report id" --rid="rec report id" --nodummy
Make sure that the caller is the owner of the resources....
output saved at ./output\1_sender.txt
Enter message to send to rec report id: hello, this is an interactive messa
ge from stdin
Enter message to send to rec report id:

(venv) C:\Data\workspace\CMP4\Projects\security>
```

Project environment

- This project is tested under windows 10 operating system
- Using Python3.8 however I believe it should work with 3.6 or higher also