Programing Language: Python version 3.9

Libraries: pycryptodome

GitHub: https://github.com/mtoczycki/CS355-Project

How to Install:

```
pip install pycryptodomex
```

Steps:

First make sure that you have installed pycryptodome, then to start either Alice or Bob go to the project folder two_way_rsa, then either go to alice_private to start Alice or bob_private to start Bob. Once you have run alice.py or bob.py with python, then you will type in a file that will be checked. The file must be placed in the private directory of Alice or Bob respectively so if Alice wants to check that her document “test.txt” is valid with Bob’s “test.txt” Alice must type in the prompt test.txt and so will Bob. If for some reason Bob or Alice stop their process in the middle, you will see that the other process may be waiting, **DO NOT PANIC!!! You will get an error “Incorrect Decryption”**. This was designed so that in case the process gets interrupted or the files are not ready at the same time, that we Bob or Alice will keep on checking to see if the file is in the public folder. Just to make things very clear that the private files, we assume and trust that neither Alice or Bob will share whatever is in the private folder and that the public folder is the only folder that they share and what others can see. If this happens and both Alice or Bob where to finish and a bit was flipped you may get this infinite loop as well. If this occurs, run the two programs again and try again from the beginning. What will be presented to Alice and Bob is a hash and it tells them if the hashes are identical or not. We also generate the hash in the beginning when you choose the file to be hashed.

Runtime

We tested the time it takes for each step in Bob’s program. Due to the way “Cryptodome” hash the files, we assumed that the longest operation will be the hashing. Through testing, hashing did significantly slow down the bigger the file got. The other steps took negligible amount of time (<= 1 seconds), therefore we will not show them here. We tested on two machines: a personal computer equipped with an SSD, and the purdue remote server: data.cs.purdue.edu

One major limitation that is worth mentioning is that the program runs for very long when hashing two large files simultaneously on an HDD. Again, this is due to the limitation of the library we used. If you are using an HDD, please do not hash the files simultaneously.

Local SSD		
	Hashing After Alice	**Hashing Simultaneously**
Small File (1KiB)	0.118 seconds	N/A (too fast)
Large File (4GiB	19.508 seconds	19.607 seconds

Remote Server: data.cs.purdue.edu		
	Hashing After Alice	**Hashing Simultaneously**
Small File (1KiB)	1.194 seconds	N/A (too fast)
Large File (4GiB	22.031 seconds	20.487 seconds

As we can see in the runtimes above, hashing a larger file will take longer in our program, but will still preform within a reasonable amount of time. In both the personal computer and the remote server, the difference is small which makes them run fairly similar. The difference could be attributed to some hardware difference or user error.

Security analysis:
Our goals are to have Alice and Bob compare each other's file without revealing their files' plaintext and adversaries could not learn anything or modify the files being transmitted.

For hiding file content, Alice and Bob agree on using the same SHA 256 hash function to turn their files into seemingly random binary content.

For preventing adversaries, Alice and Bob agree on using a hybrid encryption scheme—a combination of AES and RSA encryptions (MAC built in AES encryption function).

Assumptions:

- Alice and Bob start their conversation at an agreed time, otherwise Alice or Bob's program will wait.
- Alice and Bob communicate through an authenticated channel, meaning that Alice knows the public key and the encrypted file are from Bob. (No certificates or digital signature needed)
- Alice and Bob are honest, meaning that the file they send to others is correct and they follow the specified protocol.
- The "public" folder simulates a public environment where anyone can access. The "alice_private" and "bob_private" folders simulate Alice's and Bob's private environment respectively that can only be accessed by its owner.
- Only one encrypted file is needed to send from Alice to Bob or Bob to Alice by using the agreed SHA 256 function specified.
- AES and RSA keys are updated every round and never collide.
- Error "Incorrect Decryption" is triggered if adversaries attempt to disturb transmitted files, which notifies Alice and Bob to restart another transmission.

The scheme details are as followed: (Alice as sender, Bob as receiver)

- Alice and Bob each hashed their files using SHA 256.
- Alice and Bob each create their own RSA private key and public key pairs.
- Alice extract Bob's public key from "public" folder
- Alice creates an AES session key
- Alice encrypts her hashed file using AES encryption, and also generates an authentication tag (MAC).
- Alice encrypts the AES session key with Bob's RSA public key using RSA encryption.
- Alice places the file in the public folder.
- Bob receives Alice's file and extracts its contents, which includes encrypted data, encrypted AES key, and authentication tag.
- Bob decrypts AES session key with Bob's RSA private key.
- Bob decrypts encrypted data with AES session key, and also verifies the tag to detect unauthorized modification.
- Bob compares decrypted data with Bob's hashed file. If identical, then done.