**PART I - Use of Cryptology Tools (100 points)**

1. Which kind of asymmetric algorithms does GnuPG support?

RSA, ElGamal, DSA, ECDH, ECDSA, EdDSA

1. Comparing ElGamal and RSA, what is the main difference between both algorithms?

* Underlying assumption: RSA is eventually based on factoring (recovering p , q from n=pq) , where ElGamal is eventually based on the discrete logarithm problem in cyclic groups (recover x from h=gx ).
* Both algorithms require similar key sizes, and can (theoretically) be broken using similar approaches.
* ElGamal can be implemented using elliptic curves, drastically increasing efficiency and decreasing the required key sizes. RSA cannot be made more efficient.
* RSA is thoroughly standardized, and practically every implementation is mutually compatible.
* ElGamal has a wide range of implementations, using different representations and algebraic groups, most of which are mutually incompatible. Some of these are actually standardized.
* RSA used to be patented (expired long since);
* RSA is a deterministic algorithm.
* ElGamal is a probabilistic (randomized) algorithm.
* In RSA every key is independent (unique p and q ).
* ElGamal has a system parameter (the group) that is shared by many keys.

**For the following seven questions, the answer must type the command used to perform the task.**

1. Create at least 04 Elgamal key pairs using GnuPG and saved them into a file.

gpg --full-generate-key

1. Check if there are any other keys locally installed.

gpg –list-keys (check created key)

gpg –list-secret-keys

1. Delete the key recently generated (question 3) and reimport them into the system.

Export the key to a file (using ascii)

gpg --export-secret-keys --armor 90B946054FBBB208D84F7B1E303C40D76A638D35 > user1.sec\_key.asc

Delete the key

gpg --delete-secret-keys 90B946054FBBB208D84F7B1E303C40D76A638D35

reimport the key

gpg --import user1.sec\_key.asc

1. GnuPG implements a set of different symmetric algorithms. Compare Blowfish and AES (Cite at least three differences between then and one case where Blowfish is a better idea when comparing it with AES.

Blowfish was the direct predecessor to Twofish. Twofish was Bruce Schneier's entry into the competition that produced AES. It was judged as inferior to an entry named Rijndael, which was what became AES.

Interesting aside: at one point in the competition, all the entrants were asked to give their opinion of how the ciphers ranked. It's probably no surprise that each team picked its own entry as the best -- but every other team picked Rijndael as the second best.

That said, there are some basic differences in the basic goals of Blowfish vs. AES that can (arguably) favor Blowfish in terms of absolute security. In particular, Blowfish attempts to make a brute-force (key-exhaustion) attack difficult by making the initial key setup a fairly slow operation. For a normal user, this is of little consequence (it's still less than a millisecond) but if you're trying out millions of keys per second to break it, the difference is quite substantial.

In the end, I don't see that as a major advantage, however. I'd generally recommend AES. My next choices would probably be Serpent, MARS and Twofish in that order. Blowfish would come somewhere after those (though there are a couple of others that I'd probably recommend ahead of Blowfish).

1. Cipher a file (choose one) using AES-256, and then output is formatted in ASCII.

gpg --symmetric --armor --cipher-algo AES256 clear\_text.txt

1. Cypher and signed a file using a symmetric algorithm and saved the output file using this name: 'cypher\_signed\_file.txt.' Verify the signature and decipher the file 'cypher\_signed\_file.txt.'

gpg --sign --symmetric --armor --cipher-algo AES256 clear\_text.txt

If a signature is included in the encrypted file, GPG will automatically output the verification of the signature when it decrypts the message. You don't have to take any special action to verify the signature you just use the normal -d or --decrypt option, it happens by default.

gpg --decrypt cypher\_signed\_file.txt

output:

*gpg: AES256.CFB encrypted data*

*gpg: encrypted with 1 passphrase*

*Traditionally, haiku poems are three-line stanzas with a 5/7/5 syllable count.*

*This form of poetry also focuses on the beauty and simplicity found in nature.*

*As its popularity grew, the 5/7/5 formula has often been broken.*

*However, the focus remains the same - simple moments in life.*

*For more, take a look at these rules for writing haiku.*

*Now, let's enjoy two short samples.*

*gpg: Signature made 3/17/2021 8:43:51 AM E. South America Standard Time*

*gpg: using DSA key 90B946054FBBB208D84F7B1E303C40D76A638D35*

*gpg: Good signature from "user1 <user1@tec.mx>" [ultimate]*

1. Using an asymmetric algorithm, encrypt/decrypt a file using one of the public keys generated previously.

gpg --recipient user2@tec.mx --encrypt clear\_text.txt > clear\_text.txt.gpg

gpg --decrypt clear\_text.txt.gpg > new\_clear\_text.txt

1. Sign a message (file) after separate the signature file from the message file (detached) and verify the file using the signature.

gpg --sign clear\_text.txt

gpg --verify clear\_text.txt.gpg

gpg --output clear\_text1.txt clear\_text.txt.gpg

**PART II – Challenge (+20 points)**

Using the classes server.py and client.py and the keys generated (Part I), Implement a client x server application in Python that performs the following process.

**Reference:** <https://docs.red-dove.com/python-gnupg/>

