1. **Name** : Sharique Hussain, Email id : sharique88@ufl.edu
2. **Source files** : gatordec.c gatorenc.c makefile README.txt
3. This software contains two programs - Encryptor and Decryptor. Please give make command, this will create two sub directories Encryptor/ and Decryptor/ under current directory, these directories will have gatorenc and gatordec object files respectively. Please put input file in Encryptor/ directory.

a) When cd to Encryptor/ ,gatorenc has two possible ways of running by command line.

./gatorenc -l inputfilename.txt

./gatorenc inputfilename.txt -d 127.0.0.1:8888

First command will create the encrypted file inputfilename.txt.uf locally in current directory i.e. under Encryptor/

Second command will also send encrypted file inputfilename.txt.uf to the daemon decryptor listening on given ip and port.

Inputfilename.txt should be there inside Encryptor directory.

Assumption : In both cases please ensure that there is no file already there with name inputfilename.uf in Encryptor/ directory.

b) When cd to Decryptor/ ,gatordec has three possible ways of running by commandline.

./gatordec -l inputfilename.txt.uf (Inputfilename.txt.uf should be there inside Decryptor directory.)

./gatorenc

./gatorenc 8888

First command will generate the decrypted file in current Decryptor/ directory with filename inputfilename.txt .

Seccond command will run the gatorenc program in daemon mode and will listen at a default 127.0.0.1:5432 address.

Third command will run the daemon at 127.0.0.1:8888 address. Both second and third commands will generate decrypted file in Decryptor/ directory.

Assumption : Please ensure for first command, passed filename argument has ".uf" extension.

Please kill any gatordec process running in background before using this gatordes program using kill -9 processNumber command.

4. **Code Layout:**

a) gatorenc.c

1. Read command line arguments, create and open the output file.
2. Prompt the user for password and using that password and SHA256 hash generate a 32 byte key using function gcry_kdf_derive. Then write the random hardcoded IV in output file.
3. Using gcry_cipher_ APIs create handle, set key, set iv, and run the algorithm for AES256 encryption by reading 16 bytes of data block at a time until all file data is read. If necessary add as many '**$**' char to complete the last 16 byte block.
4. Then generate the HMAC using gcry_md_ APIs with all the data that is there in output file this includes IV in the beginning.
5 Now the output file is re-written as HMAC+VI+CIPHER and saved on the disk.
6. Now if we require to send the encrypted file over ip based on our command line arguments dflag will be true. A socket is created and connected to the daemon running at given ip and port.
7. Then send file size and file name to the server and then in a loop start sending file content in a chunk of maximum 8KB size at a time until all file data is sent.

b) gatordec.c

1. There is one decrypt method which will be called with output filename and input filename as argument. This will be used by both daemon mode and normal mode. Its flow is like, it aks for password which must be same as used for encryption. Generate a key using PKDF2 function and then generate the HMAC and compare with the extracted HMAC from input file. Then using similiar encryption algorithm except using gcry_cipher_decrypt method, we decrypt the data and save it to the output file.
2. For normal case lflag is on we just call above decrypt method, else we run the daemon at 127:0.0.1:port address. 5432 is default port or we will use the port passed in command line argument. We create socket, bind it, listen and in an infinite loop accept the connection from client to receive the data. First file size is received, then filename and then file data. Input file is created on disk and output file is also created using input file name and we then call above decrypt method.
5. PBKDF2 function : It requires passphrase, GCRY_KDF_PBKDF2 flag required when using passphrase, hash algo flag (I used SHA256 which is good), salt - I used a random 16 byte string (it needs to be same for enc and dec), iterations, and key buffer to store generated key, other arguments are there for mention buffer size in bytes. This function gives us a 32 byte key to be used with AES256 algo.
6. Number of Hours spent - approx 100 : Lots of effort was required includes system setup, studying library documentation, designing, writing and testing code etc.

I checked wth 147 Mb file and software is working fine both locally and on ip. Algorithm apppends few '$' chars at the end of file if required to make make the file size multible of 16.
I ran the test with 147 mb file and checked its time.
Median real time was 6.4s and mean was 6.7s (including time taken to read password), I have included the file generated by script command.