Phase 2 report

Requirements

Modify the online file transfer application that you had for phase 1 so that your code encrypts/decrypts the files that are exchanged between the client and the server. Use AES symmetric-key crypto system with 256-bit key and with CBC mode in your code. Use an online SHA256 to hash your student ID to generate a unique 256 bits hashed value to be used as the AES 256-bit key. You can hard code the 256-bit key in your application. As for the Initialization Vector (IV) to be used by the CBC mode, make sure to generate a new IV for each file transfer. The file sending side is responsible for generating the IV, and should send the IV to the other side. The IV should be 128 bits long to match the AES plaintext block size, and it should be generated using a cryptographically secure random number generator (RNG) function/method (research which cryptographically secure RNG function/method is the most suitable to use for your code through the Internet).
To encrypt a file to be sent to the other side, you may need to pad the file if the last generated plaintext block of the file is less than 128 bits long (research the proper way of how padding should be done through the Internet). Once the IV is generated and the encrypted file is produced, then send the IV first before sending the encrypted file to the other side (i.e., the receiving side will always expect the first 16 bytes (or 128 bits) of a received encrypted file to contain the IV that the other side has used for encrypting the file). The receiving side should use the received IV along with the hard-coded symmetric key to decrypt the received encrypted file.

Phase 2 must be your own genuine code. You may use existing libraries/methods/functions but only for AES; the rest of the code needed for phase 2 must be developed by you. Make sure to test your modified application preferably using 2 virtual machines.
Verify your code by comparing the ciphertext that your application generates against an equivalent ciphertext generated by some of the existing online AES-256-CBC tools.

Note: Some of the online AES-256-CBC tools will pad the plaintext even if it was a multiple of 128 bits while others don't. So, if your padding implementation is not similar to that of the online tool, then there will be a mismatch between only the last part of your ciphertext and the ciphertext generated by the online tool.

Implementation details

SHA256 has of my student ID `s201578750`, done using https://www.xorbin.com/tools/sha256-hash-calculator

key = `c37ddfe20d88021bc66a06706ac9fbdd0bb2dc0b043cf4d22dbbbcda086f0f48`

For the cryptographically secure random number generation, I'm using python's secrets library.

Assignment details

The code changes made for phase 2 can be found on GitHub, the branch ph2-encryption. You can see the code changes made on this page.