Homework 3
AES Encryption

For this homework assignment you will write a program to encrypt arbitrary files using the AES encryption scheme. This program, although rudimentary in terms of its interface, is intended to actually be usable.

The program, `myaesprog`, will both encrypt and decrypt, and will do so according to the following rules:

1.) The command line for this program will contain either 2 or 3 arguments. The number of arguments on the command line indicates whether `myaesprog` should "encrypt" or "decrypt". Examples:
   a. `myaesprog input_file_name encrypted_name key_file`
   b. `myaesprog encrypted_name key_file`
2.) When encoding, `myaesprog` will accept a file with an arbitrary name as its input.
   a. It only needs to handle files with names that are 59 characters or less
3.) When encoding, `myaesprog` will produce an aes encoded output file (when naming the encrypted file, I recommend that you use the suffix .aes to remind you it is an encrypted file)
4.) The encryption will use cipher block chaining mode (CBC): you MUST IMPLEMENT THIS YOURSELF and **not** use the cbc API option (this will solidify your understanding of how it works)
   a. In other words, you will use "ecb" mode and encrypt blocks of 16 bytes at a time, and then do the CBC operation yourself!
   b. You will use the python library "`cryptography`"
   c. See the sample program
5.) When decoding, `myaesprog` will accept as input a file that is known to be encrypted (again, I recommend using a .aes suffix when naming your files just to remind you)
6.) When decoding, `myaesprog` will determine the filename, decrypt the data, and write it into a file with the correct name and length
7.) Examples:
   a. To encrypt a .jpg file the invocation might look like this:
      i. `myaesprog soccer.jpg soccer.aes key_file`
         1. the program will create an output file named `socker.aes`
         2. the program will use as its encryption key the 32 bytes base64 encoded in the file `key_file`
   b. To decrypt the file encrypted above, the invocation would look like this:
      i. `myaesprog soccer.aes key_file`
         1. The program will extract the output file name (it will be able to determine this from a header we create, see below)
         2. The program will use as its decryption key the 32 bytes base64 encoded in the file `key_file`
   c. Note that the program can determine whether it is encrypting or decrypting just by looking at the number of command line arguments.

i. Implement some basic error checking:
    1. reject trying to encode if the input file has a name with more than 59 characters
    2. reject encode/decode both if the `key_file` does not contain exactly 32 bytes
    3. Optional: when decoding, check to see if the filename already exists in the current directory. If it does, insert "x_" in front of the decoded filename to prevent overwriting the already existing file
    4. Optional: reject trying to decode if the input file name doesn't end in "`.aes`" (this would force the user to follow the preferred naming convention)

8.) When encrypting, the following steps will be used:
   a. An 80 byte header will be created
      i. The first 16 bytes of the header, `hdr[0:16]`, will be a nonce created with `os.random(16)`
         1. The nonce will be encrypted and is used to initialize CBC
      ii. `hdr[16]` will be the total number of characters in the filename (uint8)
      iii. `hdr[17:76]` holds the characters of the filename as a b string (the name should be left-justified, i.e., start at hdr[17]). Python strings can be converted to a b string using `varname.encode()`, and a b string can be converted to a Python string using `varname.decode('utf-8')`
      iv. `hdr[76:80]` will encode the length of the payload in a format designed to avoid issues with internal integer representations (e.g. endianness). Note that the payload may not be a multiple of 16, but the overall `.aes` file created must be a multiple of 16. This is why we must encode the actual payload length in the header: so that when decryting we can throw away the extra bytes of padding that we need to add at the end so that the overall `.aes` file is a multiple of 16 bytes in length.
         1. Note that the maximum file length supported is 2**32 – 1 (we are using an unsigned approach to representing integers)
         2. hdr[76] is the most significant byte, while hdr[79] is the least significant byte