CPSC 526 - Assignment 4
Gregory Muchka 10153582
Matthew Powaschuk 30022573
Both in tutorial T04

Our code is a python script, no compilation necessary.

For Server.py run as "python Server.py [port] [key]"

For Client.py run as "python Client.py [command] [filename] [hostname:port] [cipher] [key]

Testing (1mb file):

Please note 2 thing about this, the file was generated using the line "dd if=/dev/zero of=mb.txt count=1024 bs=1024"
and Server.py is in the folder "tosend"

```
root@OW-THE-EDGE1:~/Desktop/a4# sha256sum mb.txt

30e14955ebf1352266dc2ff8067e68104607e750abb9d3b36582b8af909fcb58
mb.txt
root@OW-THE-EDGE1:~/Desktop/a4# python Client.py write mb.txt localhost:1234 aes256
password

Operation completed successfully!

root@OW-THE-EDGE1:~/Desktop/a4# cd tosend/

root@OW-THE-EDGE1:~/Desktop/a4/tosend# sha256sum mb.txt

30e14955ebf1352266dc2ff8067e68104607e750abb9d3b36582b8af909fcb58  mb.txt
```

Protocol:
Note that for all data transfers longer than a block, and END block is sent at the end to signify end of transfer.

This is because otherwise the program will hang waiting for the next block of data.

Also '{' is our padding character. and the last block of each transfer is padded in such a way, then followed by the
END block which is exactly "END{{{{{{{{{{{{{", which may be encrypted.

If an error occurs at any point a padded "err" block will be sent by server, and based on the state of the program the
client will be able to interpret why the err was sent.

1. First message is sent unencrypted regardless and is simply cipher and nonce

seperated by a space character.

2. Encryption key and IV are set up as required in asg description. i.e. "encr_key = hashlib.sha256(key + nonce + "SK").hexdigest()"
and "IV = hashlib.sha256(key + nonce + "IV").hexdigest()
IV = IV[:16]", respectively

3. The challenge sent by the server is a random 16 byte block, the client must compute the sha256 hash of key+nonce+challenge and send it back, encrypted unless null.

The server then confirms the hash and sends either an OK block or ERR block. Challenge is encrypted unless null.

4. The request + filename is sent seperated by a space in 16 byte blocks. The Server function getdouble()
splits this transfer based on the space, the filename can be of arbitrary length. This is sent encrypted unless null.

5. Data exchange is sent in a similar way, 16 byte blocks encrypted unless null. Last block is padded as
decribed earlier and then the END block is sent.

6.
The final success message is an "OK" block, padded and possibly encrypted unless null.

Timing:

Results:

Note: only the results under "real" are taken as returned by the time command.

1Gb:

| | aes256 | aes128 | null |
|---|---|---|---|
| first | 29m9.725s | 31m52.846s | 22m16.357s |
| second | 28m8.643s | 28m7.945s | 23m34.503s |
| third | 30m3.002s | 27m17.444s | 21m59.204s |
| median | 29m9.725s | 28m7.945s | 22m16.357s |

1Mb:

|  | aes256 | aes128 | null |
|---|---|---|---|
| first | 0m1.941s | 0m1.938s | 0m1.306s |
| second | 0m2.371s | 0m1.834s | 0m1.325s |
| third | 0m2.226s | 0m1.864s | 0m1.347s |
| median | 0m2.226s | 0m1.864s | 0m1.325s |

1Kb:

|  | aes256 | aes128 | null |
|---|---|---|---|
| first | 0m0.217s | 0m0.214s | 0m0.208s |
| second | 0m0.209s | 0m0.207s | 0m0.209s |
| third | 0m0.218s | 0m0.209s | 0m0.209s |
| median | 0m0.217s | 0m0.209s | 0m0.209s |

From these results, where Gb is the clearest on the difference between these 3 ciphers, intuitively, as AES256 has more rounds than AES128 it should take longer, and these results show that as well.

Also intuitively, not encrypting the files should be quicker than both, and these results show us as well.

From the long runtime of sending a 1gb file, especially under aes256, we see the conundrum of modern cryptography.

That is, security vs efficiency. Although the communication is more secure, in some circumstances the runtime of aes256 might be considered too long compared to the other 2 possibilities here. In instances like stock trading for example, a difference of one minute might make all the difference in a file transfer.

Note that null would be a lot faster if the file was not transfered in 16 byte blocks, its speed suffers due to this requirement.

For smaller files though, there is little reason to swap down to a lower encryption from aes256. The difference between the runtimes are very small and likely negligible in most scenarios.