**CPSC 526**

**Assignment 4**

**Tutorial: T01**

**Name: Sam Ao**

**ID: 10111222**

**Partner: Artin Rezaee-Anzabee**

**Running the program:**

**Running the server:**

Run the server by: python3 server.py <Port> <Key>

**Running the client:**

To upload a file to the server run: cat <fileToUpload> | python3 client.py <command> <fileName> <address>:<Port> <cipher> <Key>

To download from the server run: python3 client.py <command> <fileName> <address>:<Port> <cipher> <Key> > <fileToWriteTo>

Command can be read and write

Cipher can be null, aes128, and aes256

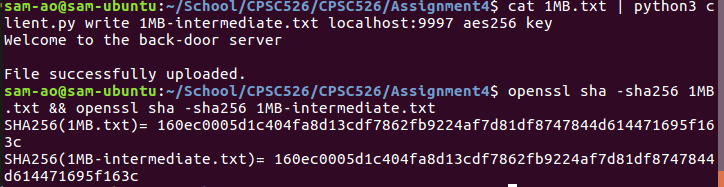
Key is the key used by server and client for authentication. The key used by server and client must be the same for the communication to progress

**Connecting to server and handshake details:**

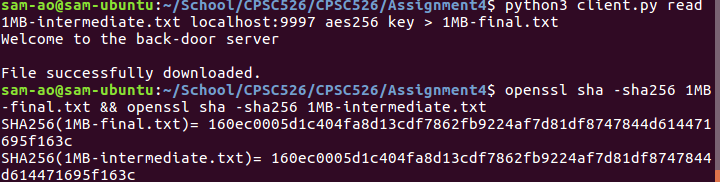
The server will be bound to a source port and will accept clients. Upon client’s connection, server received an unencrypted message from the client including the nonce and cipher specified by the client. The server will then use this information to create the initialization vector and session key for this particular session. Server will then use the specified cipher (or null) to encrypt further communication between the client. The server will then challenge the client about its knowledge of the key. To do so, the server generates a random authentication token and sends it to the client. The client then has to add their key to the authentication token and send the hash of the auth token and key over to the server. The server then does the same and if the two strings match, the server allows the client to execute its command (read/write). Each client is only able to execute one command before they are terminated. If the client is uploading a file, the server will receive data from client, 16 bytes at a time if it is encrypted, and write the received data to a file specified by the user. Once all the data is sent, the client will inform the server that it is completed sending data. If the command is read, then the server will read from the file specified by the user and send each line to the client, 16 bytes at a time if it is encrypted. If the user indicated that the data must be encrypted, before messages are sent from the client or from the server, they will break the data into 16 byte chunks, and pad any data that is less than that using PCKS7. Then the data chunks are encrypted using AES128 or AES256, before being sent to their respective destinations. Once they arrive their destinations, they are decrypted, and then unpadded. The server will send a feedback regarding the success or failure of the operation to the client. When the operation is over, the server will the terminate the client.

**Client Server operation**

**Uploading a 1MB file using AES256**

****

**Downloading a 1MB file using AES256**

****

**Timing**

**WRITE:**

1KB

null:

|  |
| --- |
| 0.179 |
| 0.322 |
| 0.180 |
| 0.191 |
| 0.288 |
| 0.202 |
| 0.194 |
| 0.243 |
| 0.184 |
| 0.188 |

Avg: 0.2171

Median: 0.191

Aes128:

|  |
| --- |
| 0.695 |
| 0.507 |
| 0.537 |
| 0.522 |
| 0.515 |
| 0.515 |
| 0.508 |
| 0.516 |
| 0.524 |
| 0.506 |

Avg: 0.5345

Median: 0.516

Aes256:

|  |
| --- |
| 0.500 |
| 0.477 |
| 0.483 |
| 0.477 |
| 0.511 |
| 0.504 |
| 0.507 |
| 0.483 |
| 0.479 |
| 0.484 |

Avg: 0.4905

Median: 0.483

1MB

null:

|  |
| --- |
| 0.206 |
| 0.218 |
| 0.204 |
| 0.208 |
| 0.212 |
| 0.204 |
| 0.208 |
| 0.202 |
| 0.206 |
| 0.207 |

Avg: 0.2075

Median: 0.206

Aes128:

|  |
| --- |
| 2.768 |
| 2.802 |
| 2.827 |
| 2.805 |
| 2.815 |
| 2.850 |
| 2.779 |
| 2.886 |
| 2.801 |
| 2.816 |

Avg: 2.8149

Median: 2.805

Aes256:

|  |
| --- |
| 2.795 |
| 2.786 |
| 2.786 |
| 2.766 |
| 2.861 |
| 2.769 |
| 2.784 |
| 2.8 |
| 2.950 |
| 2.911 |

Avg: 2.8208

Median: 2.786

1GB

null:

|  |
| --- |
| 72.356 |
| 76.488 |
| 87.946 |
| 83.909 |

Avg: 80.17475

Median: 76.488

Aes128:

|  |
| --- |
| 2911.187 |
| 2807.459 |
| 2998.548 |
| 2976.823 |

Avg: 2928.5035

Median: 2911.187

Aes256

|  |
| --- |
| 2946.528 |
| 2899.645 |
| 2953.62 |
| 2921.127 |

Avg: 2930.23

Median: 2921.127

**READ:**

1KB

null:

|  |
| --- |
| 0.178 |
| 0.160 |
| 0.148 |
| 0.146 |
| 0.149 |
| 0.148 |
| 0.149 |
| 0.156 |
| 0.146 |
| 0.146 |

Avg: 0.2171

Median: 0.148

Aes128:

|  |
| --- |
| 0.715 |
| 0.415 |
| 0.399 |
| 0.415 |
| 0.433 |
| 0.433 |
| 0.428 |
| 0.429 |
| 0.429 |
| 0.441 |

Avg: 0.5345

Median: 0.433

Aes256:

|  |
| --- |
| 0.426 |
| 0.407 |
| 0.419 |
| 0.412 |
| 0.413 |
| 0.412 |
| 0.413 |
| 0.405 |
| 0.411 |
| 0.415 |

Avg: 0.4905

Median: 0.413

1MB

null:

|  |
| --- |
| 0.189 |
| 0.171 |
| 0.172 |
| 0.173 |
| 0.169 |
| 0.176 |
| 0.161 |
| 0.172 |
| 0.165 |
| 0.159 |

Avg: 0.2075

Median: 0.176

Aes128:

|  |
| --- |
| 2.835 |
| 2.886 |
| 3.075 |
| 2.964 |
| 2.846 |
| 2.857 |
| 2.821 |
| 2.867 |
| 2.867 |
| 2.825 |

Avg: 2.8149

Median: 2.857

Aes256:

|  |
| --- |
| 2.942 |
| 2.868 |
| 2.936 |
| 2.811 |
| 2.834 |
| 2.898 |
| 2.827 |
| 2.892 |
| 2.891 |
| 2.999 |

Avg: 2.8208

Median: 2.891

1GB

null:

|  |
| --- |
| 71.686 |
| 77.381 |
| 85.90 |
| 86.952 |

Avg: 80.17475

Median: 77.381

Aes128:

|  |
| --- |
| 2961.227 |
| 2826.452 |
| 2878.162 |
| 2982.661 |

Avg: 2928.5035

Median: 2878.162

Aes256

|  |
| --- |
| 2996.751 |
| 2977.545 |
| 2992.601 |
| 2945.158 |

Avg: 2930.23

Median: 2977.545

When uploading small files, the performance difference between unencrypted data transfer and encrypted data transfer is small. On average, it only takes a little over twice as well long to transfer encrypted data as it does to transfer unencrypted data, when uploading a 1KB file. However, with a larger file size of 1MB, it takes 14 times as long to transfer encrypted data compared to unencrypted data. When uploading a 1GB file this difference is even more apparent. On average it takes 36 times as long to upload encrypted data as it does to upload unencrypted data. The results for the read function proved to be very similar. From this we can see that encryption is a significant performance drain, especially when handling larger files.