CPSC 526 Network Security
Tyrone Lagore & James MacIsaac

Time Results

Tyrone Lagore & James MacIsaac

The results of our tests show that as the size of the file increases, so does the effect that encrypting them has on the time it takes to transfer the file. In general, no encryption always transfers the quickest, showing only slight increases on smaller data, but over a 255% increase in time on AES128 for a 1GB file, and a 270% increase in time using AES256. What this leads us to believe, is that the added security in using AES256 over AES128 far outweighs its increased cost in time. There was only approximately a 7-10% increase in time from switching from AES128 to AES256. However, obviously sending with no encryption yields the quickest results. Our results have lead us to conclude that if data does not have a need to be encrypted, it is a large time loss to perform the encryption. However, if you are going to encrypt your data, you should use AES256. (If AES128 and AES256 are your only choices)

Running the Program and Protocol Description

cpsc_526_assignment3 - netsec_secure_ftp

Authors:
Tyrone Lagore T01 (10151950) James MacIsaac T03 (10063078)

Description:
Network data transfer system that uses AES encrypted communications. Contains a client and a server application.

Running the program
open the package containing the files.
The client and server should be ran in separate folders, as they are organized in the distribution.
We require you to have PyCrypto and python3 to be installed in order to run the program.

Setting up the venv

To set up an environment to install Pycrypto follow these steps:

1) install virtualenv using the command
   'pip3 install virtualenv'
2) create a virtualenv called 'venv' in the project root folder by using the command
   'python3 -m venv venv'
3) activate the venv using
   'source venv/bin/activate'
4) install PyCrypto using
   pip3 install pycrypto

CPSC 526 Network Security
Tyrone Lagore & James MacIsaac

You now have a vitualenv with pycrypto ready to run the program.

Running the server:
enter the 'server' folder that is in the project root folder.
start the server using this command and argument scheme:

    python3 __main__.py <port> [key]

where
    port is the port to listen on
    key is the optional secret key to use for encrypted communications

The server will display it's public facing IP to allow for easy startup of the client.

Running the client:
enter the 'client' folder that is in the project root folder.
start the client using this command and argument scheme

    python3 __main__.py <read|write> <filename> <host>:<port> <none|aes128|aes256> [key]

where
    mode can be read or write
    filename is the name of the file to be read/written
    host is the host ip that is running the server
    port is the port on the host machine to communicate with the server
    encryption scheme for communications is off/aes128/aes256
    key to use for encryption (not used if no encryption is to happen)

Test Output:

Test ran is a read of a picture file using aes256 encryption and secret key: notsosecret123

Server side

!! 19:06:18: ------------------------------------
!! 19:06:18: Listening @ 172.19.1.45 on port 8888
!! 19:06:18: Using secret key: notsosecret123
!! 19:06:18: ------------------------------------

!! 19:06:26: ------------------------------------
!! 19:06:26: Client connected: 172.19.1.45
!! 19:06:26: Cipher: aes256
!! 19:06:26: IV: b'?\\\x13\xfa\xcd\xd2f\x81Yq2\xe7\n=^J'

!! 19:06:26: Sending challenge...
!! 19:06:26: Received good response. Initializing communication.
!! 19:06:26: Client requesting filename: Blue_square_X.PNG
!! 19:06:26: Processing...
!! 19:06:26: 100.0%
!! 19:06:26: File sent, checksum: 74feef5a31fc985247b8964c09e2433b
!! 19:06:26: Finished sending file.
!! 19:06:26: Done
!! 19:06:26: ----------

Client side

!! Client starting. Arguments:
!! host: 172.19.1.45
!! port: 8888
!! command: read
!! filename: Blue_square_X.PNG
!! cipher: aes256
!! key: notsosecret123
!! Receiving file...
!! File confirmed, checksum: 74feef5a31fc985247b8964c09e2433b

Communication protocol:

Our communication protocol functions by controlling the overall interaction between the client and the server using a synchronous request-response message-based system for encrypted communication

Connection establishment:
The server checks that the client has connected and spawns a thread to handle it

Client authenticity (handshake process):
The client sends the server an unencrypted message containing an initialization vector and a chosen cipher.

The server does not trust the client yet - it generates a nonce, encrypts it using the client chosen cipher and IV along with the server-known secret key. This is then sent to the client.

The client receives the encrypted nonce, decrypts it, adds 1 to the decrypted value, then re-encrypts it, and sends the re-encrypted value to the server for verification.

The server decrypts the client response, checks that it is the original nonce + 1, and then either continues communication (client responded well) or disconnects the socket (client was wrong, cannot prove that they are authentic).

Encrypted communication:
Once verified, the encrypted communication can start.

The sender will get the contents of the file ready. These contents are split into fixed size 'message' objects. These message objects are serialized. The serialized data is then appended to a header object, simply indicating the length of the serialized message data.

These header & serialized message objects are padded, and individually encrypted using the cipher, IV, and key.

The encrypted messages are then sent over the communication channel in order.

The receiver decrypts the messages, acquires the message length in the header object, reads the valid data from the serialized message object (knows when the data ends and the padding begins thanks to the header), then deserializes the message and interprets it.

Once all data has been received in this manner, the data is now useable.