**SECURE CLIENT-SERVER FILE TRANSFER: SECURE TCP FILE TRANSFER USING EKE PROTOCOL**

EE209 Project

Fall 2015

By

Yash Adukia - 009988602

Pooshan Vyas - 010034362

# 

Department of Electrical Engineering

San Jose State University

San Jose, CA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Introduction**

The purpose of the project is to implement a secure file transfer algorithm in a TCP client server environment. The major services provided by this implementation are:

* Username and password authentication
* Mutual authentication of client and server
* Confidentiality of the transmitted file at the client
* Integrity of the file which received by the server

**Working**

The security of the protocol is based on the EKE (Encrypted Key Exchange) protocol feature. This works by transmitting the file from server to the client only after the server and client are mutually authenticated. The algorithm uses Diffie Hellman key exchange to mutually authenticate the client and the server and then the server transfers the file to the user (client).

**Implementation**

The code to make this project work with all the above listed features was done in Python.

Starting steps as implementation:

**Client Step 1 – Message 1**

* The client is prompted to input a username and then password. Both things are sent to the server to get authentication approval.
* After that, a client secret random number (Xa) is chosen. We are given with the generator (g) and the prime modulus (p).
* We find the mod using the formula, mod = g^Xa mod p
* Now, we XOR the mod calculated in the previous step with the user password.
* This XOR is then sent to the server along with p, g and ClientID (username)

**Server Step 1 – Message 2**

* The server receives p, g, client ID and XOR
* Decrypts the XOR and gets g^Xa
* Server choses its own secret random number, Xs
* Server calculates mod = g^Xs mod p
* Just like the client, server also XOR’s the mod with the user password
* Server then calculates the Key, Kas = g^XaXs mod p which is the sahred secret key
* A nonce Ns is randomly generated at the server which is equal to the AES block sixe, that is, a multiple of 16.
* The nonce Ns is then encrypted using AES with the key Kas
* Now, the XOR and AES ciphertext is sent to the client

**Client Step 2 – Message 3**

* The client decrypts XOR from server just like the server did before
* It gets g^Xs and calculates the secret shared key
* Kas = g^XaXs mod p should match on both server and client
* AES ciphertext is then decrypted and nonce Ns is retrieved.
* Client takes its own nonce Na and concatenates it with Ns
* AES ciphertext is generated using Kas and Na||Ns

**Server Step 2 – Message 4**

* Server decrypts the AES sent by the client and retrieves Na, client nonce
* Server then uses AES to encrypt Na using shared key Kas and sends it to client

**Server Step 3 – Message 5**

* Now, the server takes a file as input and encrypts the file using the SHA-1 algorithm
* Generates a Message Digest (MD), which is nothing but the hash for the file
* The encrypted file and the hash is sent to the client

**Client Step 3**

* The client receives the file and calculates a hash of the file using SHA-1 algorithm
* Now, the server hash and the client hash is compared
* If they match, then the integrity and the confidentiality of the file is intact
* If not, then Trudy has modified the content or was able to extract the file

**Result**

The file transfer and the key generation was done as expected and explained above.

* The shared key Kas was matching at server and client
* The file was received by the client and the hash was found out to be the same. Hence, we conclude that the integrity of the file has been intact
* There was no modification made to the file

Therefore, the project was successfully completed and the expected results were obtained.

**Python Code**

**TCP Server:**

import socket

import doctest

from Crypto.Cipher import AES

import random

import sys

import struct

import hashlib

import os

def Main():

host = 'localhost'

# port = raw\_input("Enter port number: ")

# port = int(port)

port = 4000

s = socket.socket()

s.bind((host, port))

s.listen(1)

c, addr = s.accept()

print "Connection from: " + str(addr)

while True:

data = c.recv(1024)

if not data:

print "----- END OF SESSION, BYE! ------"

break

print '\n ------------------ Step 1 ----------------------- '

print "from connected user: " + str(data)

print '\n ------------Username and Password----------------'

if str(data) == 'yash101' or 'pooshan101':

client\_id = str(data)

print client\_id # getting client ID

c.send('u\_ack')

password = c.recv(1024)

if str(password) == 'niceday':

c.send('Authenticated')

p = c.recv(1024) # getting p

c.send('pAck')

print " P received "

p = int(p)

g = c.recv(1024) # getting g

c.send('gAck')

print "G received"

g = int(g)

xored\_client = c.recv(1024) # getting xor of client

c.send('xorAck')

print "Xor client received"

# xored\_client = xored\_client

print '\n ------ Value of P : --------', p, g,

print '\n ------ Value of g : --------', g

print '\n ------ Value of xored\_client : --------', xored\_client

# server\_secret = 197443274309234702374320493274939487257

server\_secret = 5516

password = password.ljust(39, '0')

print ' Password: ', password

print '\n ------------------ Step 2 ----------------------- '

# ---------------MOD STARTS--------------

mod\_serverDH = modlargeNum(g, server\_secret, p)

print "mod\_serverDH g^Xs mod p : ", mod\_serverDH

# ---------------MOD STARTS--------------

# ---------------XOR STARTS--------------

xorWithPass = xor\_message(str(mod\_serverDH), password)

print "XORed with Password : ", xorWithPass # returns char - M2 SEND TO CLIENT

xored\_hex\_server = "".join("{:02x}".format(ord(c)) for c in xorWithPass)

print 'xored\_hex\_server', xored\_hex\_server # returns HEX

# ---------------XOR ENDS--------------

print ' \n ---------------XOR DECRYPT and ENCRYPT-------------- '

decryptClientXor = xor\_message(xored\_client, password)

print "dec client XOR G^xa: ", decryptClientXor #returns CHAR

encryptAgain = xor\_message(decryptClientXor, password)

print "Match with client XOR : ", encryptAgain #returns CHAR

print "Key Kas is g^(XaXs) mod p"

keyKas = modlargeNum(int(decryptClientXor), server\_secret, p)

print " ---------------- Kas KEY : --------------", keyKas #returns INT

# ---------------ENDS--------------

# nonce\_Ns = generate\_nonce() # returns STR

nonce\_Ns = '11100111011100111000000000000000'

print "Server NONCE : ", nonce\_Ns

print "Nonce is the text for AES", type(nonce\_Ns)

print '\n ------------------ Step 3 ----------------------- '

print '\n ---------------AES STARTS-------------- '

# KEY and TEXT have to be STR format to perform AES

key = '49327493294327478947847328894738' # 16 byte key for AES(128)

print '\n --- Kas match both side client and server, hence we perform the right opration and it is correct result ---\n '

newKas = str(keyKas)[:32]

# newKas = int(newKas)

# print "NEW KAS", (newKas)

# print sys.getsizeof(key)

# print sys.getsizeof('82395155117150893193249167212321992151')

# key = '42394503760154450521289873942225720466'

print "Key in DECIMAL", int(newKas)

IV = 16 \* '\x00' # Initialization vector: discussed later

mode = AES.MODE\_CBC

encryptor = AES.new(newKas, mode, IV=IV)

# text below is nothing but server\_secret in string format

# text = '3476576834593040621903216239480246234712872104970004097324072013'

ciphertext = encryptor.encrypt(nonce\_Ns)

print "AES Cipher Text: ", ciphertext #returns CHAR - M2 SEND TO CLIENT

# print "in HEX : ", "".join("{:02x}".format(ord(c)) for c in ciphertext) #CHAR to HEX

# print int("".join("{:02x}".format(ord(c)) for c in ciphertext), 16) #Hex to Dec

# ---------------AES ENDS--------------

print 'sizeof(xorWithPass)', sys.getsizeof(xorWithPass)

print 'sizeof(ciphertext)', sys.getsizeof(ciphertext)

# c.send('message2')

c.sendall(xorWithPass)

xorConf = c.recv(1024)

if xorConf == 'xorServerAck':

print "Server XOR Sent"

c.sendall(ciphertext)

aesConf = c.recv(1024)

if aesConf == 'aesAck':

print "XOR and AES transferred successfully"

print '\n ------------------ Step 4 ----------------------- '

# ---------------------------------------------------

# ----------------MESSAGE 2 ENDS HERE----------------

# ---------------------------------------------------

aes\_client\_ciphertext = c.recv(1024)

c.send('clientAesAck')

print "aes\_client\_ciphertext received: ", aes\_client\_ciphertext

# ---------------------------------------------------

# ----------------MESSAGE 3 ENDS HERE----------------

# ---------------------------------------------------

# ------------AES DECRYPTION STARTS-----------

# below was received from client

decryptor = AES.new(newKas, mode, IV=IV)

concatenatedNonceFromClient = decryptor.decrypt(aes\_client\_ciphertext)

print 'concatenatedNonceFromClient', concatenatedNonceFromClient

# ------------AES DECRYPTION ENDS-----------

#Splitting the Nonce Na||Ns done at Client

#Retrieving Nonce Na

split\_nonce\_Na = concatenatedNonceFromClient[:16]

print "Split Recovered Nonce Na : ", split\_nonce\_Na

# ------------AES ENCRYPTION STARTS-----------

encryptor = AES.new(newKas, mode, IV=IV)

ciphertext\_Na = encryptor.encrypt(split\_nonce\_Na)

print "AES Na cipher at Server: ", ciphertext\_Na

# ------------AES ENCRYPTION ENDS-----------

# ------Sending the Nonce Na ciphertext to Client-----

print '\n ------------------ Step 4 ----------------------- '

clear\_to\_send\_Na\_AES = c.recv(1024)

if clear\_to\_send\_Na\_AES == 'send\_aes\_nonceNa':

c.sendall(ciphertext\_Na)

# aesServerNonceNaConf = c.recv(1024)

# if aesServerNonceNaConf == 'aesNonceNaAck':

print "Message 4 Transfer Success"

# ---------------------------------------------------

# ----------------MESSAGE 4 ENDS HERE----------------

# ---------------------------------------------------

# -----------------MESSAGE 5 START----------------------------

print ' \n -------------Secure FILE TRANSFER STARTS---------------'

print '\n ------------------ Step 5 ----------------------- '

fileSentCount = 0

if fileSentCount < 3:

if os.path.exists('testfile1.pdf'):

length = os.path.getsize('testfile1.pdf') # get file size in bytes

c.send(str(length)) # has to be 4 bytes

len\_sent = c.recv(1024)

if len\_sent == 'LnACK':

c.sendall('ok')

filename = 'testfile1.pdf'

f = open(filename, 'rb')

l = f.read(1024)

while (l):

c.sendall(l)

#print('Sent ', repr(l))

l = f.read(1024)

f.close()

fileSentCount += 1

print 'File sent!'

# -------------FILE TRANSFER ENDS------------------

# -------------ADDITIONAL FILE TRANSFER REQUEST----

confi = c.recv(1024)

if confi == 'File delivered':

print 'File delivered'

elif fileSentCount < 3:

if os.path.exists('testfile1.pdf'):

length = os.path.getsize('testfile1.pdf') # get file size in bytes

c.send(str(length)) # has to be 4 bytes

filename = 'testfile1.pdf'

f = open(filename, 'rb')

l = f.read(1024)

while (l):

c.sendall(l)

print('Sent ', repr(l))

l = f.read(1024)

f.close()

fileSentCount += 1

print 'File sent!'

else:

print 'Sorry!, You have reached Max limit to request same file. '

print '\n -------------SHA1 START------------------------'

def sha1ofFile(helpMe):

sha = hashlib.sha1()

with open(helpMe, 'rb') as f:

while True:

block = f.read(2 \*\* 10) # Magic number: one-megabyte blocks (1 MB = 1024).

if not block: break

sha.update(block)

# print sha.hexdigest()

return sha.hexdigest()

# -------------SHA1 ENDS--------------------------

sha1ofFile('testfile1.pdf')

print 'SHA1 of server file is: ', sha1ofFile('testfile1.pdf')

fileSHA1 = sha1ofFile('testfile1.pdf')

c.send(fileSHA1)

else:

c.send('Not Authenticated')

else:

c.send('Wrong Username')

# """------------Username and Password ENDS----------------"""

c.close()

# --------------------FUCTIONS / METHODS ------------------------------------

def send\_msg(sock, msg):

# Prefix each message with a 4-byte length (network byte order)

msg = struct.pack('>I', len(msg)) + msg

sock.sendall(msg)

def generate\_nonce(length=16):

# """Generate pseudorandom number."""

return ''.join([str(random.randint(0, 9)) for i in range(length)])

def modlargeNum(base,power,p):

if power ==0:

return 1

if power % 2 ==0:

tmp=modlargeNum(base,power/2,p)

return (tmp \* tmp) % p

else:

return (base \* modlargeNum(base,power-1,p)) % p

def xor\_message(a, b): # xor two strings of different lengths

if len(a) > len(b):

return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])

else:

return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])

def convert\_to\_bytes(no):

result = bytearray()

result.append(no & 255)

for i in range(3):

no = no >> 8

result.append(no & 255)

return result

# -------------------Main()-------------------

if \_\_name\_\_ == '\_\_main\_\_':

doctest.testmod(verbose = True)

Main()

**TCP Client:**

import socket

import doctest

from itertools import izip, cycle

import itertools

import base64

import binascii

import struct

import sys

from Crypto.Cipher import AES

import random

import hashlib

import os

def Main():

host = 'localhost'

# port = raw\_input("Enter port number: ")

# port = int(port)

port = 4000

s = socket.socket()

s.connect((host, port))

count = 0

#DH PARAMETERS

p = 197221152031991558322935568090317202983

g = 2

# client\_secret = 197221152031991558322935568097826974661

client\_secret = 4597

print '\n ------------------ Step 1 ----------------------- '

message = raw\_input("Enter Username->")

while message != 'q':

s.send(message)

data = s.recv(1024)

print "Received from server: " + str(data)

if str(data) == 'u\_ack':

password = raw\_input("Enter Password ->")

print password

s.send(password)

decision = s.recv(1024)

if str(decision) == 'Authenticated':

print "User Authenticated"

# mod\_1 = (g\*\*client\_secret) % p

# mod\_1 = (2\*\*4) % 6 #for testing small numbers

mod\_1 = modlargeNum(g, client\_secret, p)

print "mod 1 : ",mod\_1

print 'sizeof(mod\_1)', sys.getsizeof(mod\_1)

print '\n ------------------ Step 2 ----------------------- '

print "----------Starting the real SECURE Communication-----------"

# ---------------XOR STARTS--------------

# xor\_1 = strxor(str(mod\_1), password)

password = password.ljust(39, '0')

print 'password', password

#XOR converts STRING to DECIMAL and returns CHAR

xor\_test = xor\_message(str(mod\_1), password)

# pw\_bin = ' '.join(format(ord(x), 'b') for x in password)

# print pw\_bin, mod\_1

# ---------------XOR ENDS--------------

print "XORed message: ", xor\_test

# ------------TESTING SHIT STARTS---------------

# new\_xor = xor\_strings(str(mod\_1), password)

# print "NEW XOR message: ", new\_xor

# xor\_decrypt = xor\_message(xor\_test, password)

# print "XOR decrypt : ", xor\_decrypt

# ------------TESTING SHIT ENDS---------------

#-------FROM CHR TO HEX BELOW

# xored\_hex = "".join("{:02x}".format(ord(c)) for c in xor\_test)

# print xored\_hex

s.sendall(str(p)) # sending p to server

pConf = s.recv(1024)

if pConf == 'pAck':

print "P sent"

s.sendall(str(g)) # sending g to server

gConf = s.recv(1024)

if gConf == 'gAck':

print "g sent"

s.sendall(str(xor\_test))

xorConf = s.recv(1024)

if xorConf == 'xorAck':

print "Client XOR Transferred"

print "P G and XOR message sent to server"

print '\n ------------------ Step 3 ----------------------- '

# s.send(str(xored\_hex)) # sending xored hex to server

# ---------------------------------------------------

# ----------------MESSAGE 1 ENDS HERE----------------

# ---------------------------------------------------

# --------------From Server Starts MESSAGE 2-------------

# while 1:

# xored\_server = s.recv(1024)

# if not xored\_server:

# break

# if not aes\_server\_cipertext:

# break

xored\_server = s.recv(1024)

s.sendall('xorServerAck')

print "xor server received"

aes\_server\_cipertext = s.recv(1024)

s.sendall('aesAck')

print "aes server received"

# xored\_server = recv\_msg(s)

# aes\_server\_cipertext = recv\_msg(s)

print 'xored\_server', xored\_server

print "AES: ", aes\_server\_cipertext

# --------------From Server Ends---------------

decryptServerXor = xor\_message(xored\_server, password)

print "Dec server XOR g^(Xs) : ", decryptServerXor

encryptServerXor = xor\_message(decryptServerXor, password)

print "Match with Server XOR : ", encryptServerXor

decryptServerXor = int(decryptServerXor)

keyKas = modlargeNum(decryptServerXor, client\_secret, p)

print "\n ------------------ Kas Key : ---------------------- ", keyKas #returns INT

newKas = str(keyKas)[:32]

print '\n newKas', newKas

print '\n -- Kas match both side client and server, hence we perform the right opration and it is correct result ---\n '

# ---------------------------------------------------

# ----------------MESSAGE 2 ENDS HERE----------------

# ---------------------------------------------------

print '\n ------------------ Step 4 ----------------------- '

print ' \n------------AES DECRYPTION STARTS-----------'

IV = 16 \* '\x00' # Initialization vector: discussed later

mode = AES.MODE\_CBC

decryptor = AES.new(newKas, mode, IV=IV)

nonce\_Ns = decryptor.decrypt(aes\_server\_cipertext)

print nonce\_Ns, type(nonce\_Ns)

# ------------AES DECRYPTION ENDS-----------

# Generating Nonce Na below

# nonce\_Na = generate\_nonce() # returns STR

nonce\_Na = '11000110001100011000000000000000'

#print "Client NONCE Na : ", nonce\_Na

print "Nonce is the text for AES"

# Concatenationg Nonces, Na and Ns

concatenationNonce = nonce\_Na + nonce\_Ns

#print "Nonce concatenation Na||Ns : ", concatenationNonce

# ------------AES ENCRYPTION STARTS-----------

encryptor = AES.new(newKas, mode, IV=IV)

ciphertext = encryptor.encrypt(concatenationNonce)

#print "AES Cipher Text at Client: ", ciphertext

# ------------AES ENCRYPTION ENDS-----------

s.sendall(ciphertext)

aesClientConf = s.recv(1024)

if aesClientConf == 'clientAesAck':

print "Message 3 Transfer Success"

# ---------------------------------------------------

# ----------------MESSAGE 3 ENDS HERE----------------

# ---------------------------------------------------

s.sendall('send\_aes\_nonceNa')

aes\_nonce\_Na\_cipher = s.recv(1024)

# s.sendall('aesNonceNaAck')

#print "AES of Nonce Na from Server : ", aes\_nonce\_Na\_cipher

print "Message 4 Received"

# ---------------------------------------------------

# ----------------MESSAGE 4 ENDS HERE----------------

# ---------------------------------------------------

print '\n ------------------ Step 5 ----------------------- '

# -----------------MESSAGE 5 START----------------------------

print ' \n -------------Secure FILE TRANSFER STARTS---------------'

# length = s.recv(4)

length = s.recv(16)

s.sendall('LnACK')

size = int(length)

current\_size = 0

buffer = b""

ctsFile = s.recv(1024)

if ctsFile == 'ok':

while current\_size < size:

data = s.recv(1024)

if not data:

break

if len(data) + current\_size > size:

data = data[:size - current\_size] # trim additional data

buffer += data

# you can stream here to disk

current\_size += len(data)

# you have entire file in memory

#print '\n ------- File received from Server, START HERE-------- \n \n ', buffer

print '\n ------- File received from Server, END HERE-------- '

print '\n ------ File Successfully Received ------- '

s.send('File delivered')

# -------------FILE TRANSFER ENDS----------------

# -------------SHA1 START------------------------

hash\_object = hashlib.sha1(buffer)

hex\_dig = hash\_object.hexdigest()

print '\n Client SHA1 of this file is: ', hex\_dig

serSHA1 = s.recv(1024)

print '\n Server SHA1 of this file is: ', serSHA1

# -------------SHA1 ENDS--------------------------

if hex\_dig == serSHA1:

print ' \n ------ Integrity report: File integrity is intact :) ------ '

else:

print ' \n ----- Hey! CAUTION!: File is corrupt and may be altered. ----- '

print ' Advise: request new file --- you have max 3 attempts to get file '

break

else:

print "User NOT Authenticated"

count += 1

# password = raw\_input("Try Again ->")

else:

print "WRONG username..."

count += 1

if count < 3:

message = raw\_input("Try Again Username ->")

if count == 3:

print "ACCESS DENIED"

break

# ------------Username and Password ENDS----------------

s.close()

# --------------------FUCTIONS / METHODS ------------------------------------

def generate\_nonce(length=16):

# """Generate pseudorandom number."""

return ''.join([str(random.randint(0, 9)) for i in range(length)])

def recv\_msg(sock):

# Read message length and unpack it into an integer

raw\_msglen = recvall(sock, 4)

if not raw\_msglen:

return None

msglen = struct.unpack('>I', raw\_msglen)[0]

# Read the message data

return recvall(sock, msglen)

def recvall(sock, n):

# Helper function to recv n bytes or return None if EOF is hit

data = ''

while len(data) < n:

packet = sock.recv(n - len(data))

if not packet:

return None

data += packet

return data

def xor\_strings(s,t):

"""xor two strings together"""

return "".join(chr(ord(a)^ord(b)) for a,b in zip(s,t))

def modlargeNum(base,power,p):

if power ==0:

return 1

if power % 2 ==0:

tmp=modlargeNum(base,power/2,p)

return (tmp \* tmp) % p

else:

return (base \* modlargeNum(base,power-1,p)) % p

def xor\_message(a, b): # xor two strings of different lengths

if len(a) > len(b):

return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])

else:

return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])

def recvFunc(self, msgLen):

msg = ""

bytesRcvd = 0

while bytesRcvd < msgLen:

chunk = self.s.recv(msgLen - bytesRcvd)

if chunk == "": break

bytesRcvd += len(chunk)

msg += chunk

if string.find(msg, "\n"): break

return msg

# -------------------Main()-------------------

if \_\_name\_\_ == '\_\_main\_\_':

doctest.testmod(verbose = True)

Main()