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 =
19744327430923470237432049327493948725 7
                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 =
'34765768345930406219032162394802462347128721049700040973240720l3'
                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 = 19722115203199155832293556809031720298 3
    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 = '1100011000110001100000000000000'
                #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()
```