## CSC574 Project 2

**Email Security**

**Fall 2015**

**Email Security**

**Submitted By:**

**Purna Mani Kumar G - 200066404**

**&**

**Rashmi Sandilya - 200084902**

**Goal 1:**

The Project\_2 folder contains folder Goal1. Inside the Goal1, there are 2 files outfile.txt and script.sh

The script.sh is shell script file and can be executed as below to find the secret key and secret message.

Command: ./script.sh

**Contents of script.sh**

#!/bin/bash

clear

rm -f Probable.txt

rm -f Final.txt

rm -f file.txt

rm -f new\_file.txt

for n in {a..z}{a..z}{a..z};do

echo -n " $n";

openssl enc -des-cbc -base64 -d -in outfile.txt -k $n -out file.txt

if [ $? -eq 0 ] ; then

echo "Probable"

echo "$n" >> Probable.txt

else

echo "Not Probable"

fi

done

echo "Probables are :"

while read line; do

echo $line

rm -f file.txt

rm -f new\_file.txt

openssl enc -des-cbc -base64 -d -in outfile.txt -k $line -out file.txt

tr -cd '\11\12\15\40-\176' < file.txt > new\_file.txt

if cmp file.txt new\_file.txt >/dev/null 2>&1; then

echo "$line" >> Final.txt

fi

done < Probable.txt

while read line; do

echo "Secret Password is $line"

rm -f file.txt

rm -f new\_file.txt

openssl enc -des-cbc -base64 -d -in outfile.txt -k $line -out file.txt

echo "Secret Message is "

cat file.txt

done < Final.txt

When script.sh is executed, for all the combinations of 3 letter password, the file is decrypted. If decrypting the file with password does not give error, this key is stored in Probable.txt. This file contains all the probable keys which did not respond to error while decrypting.

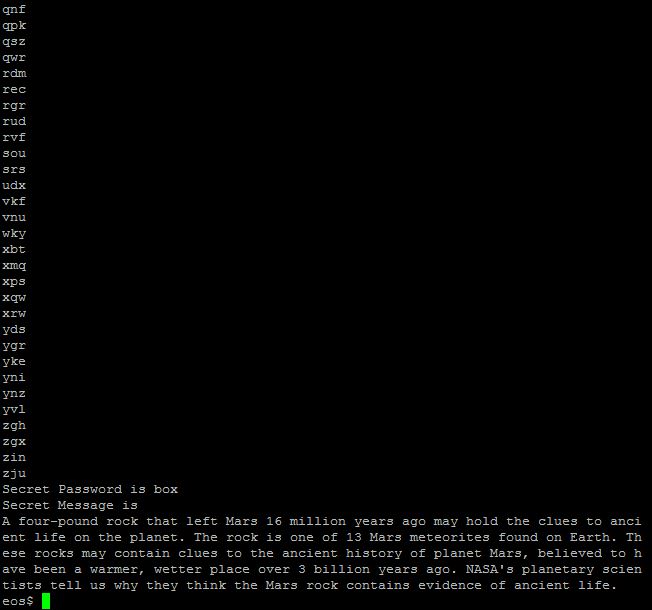
In second stage, all the probable keys are again used to decrypt the outfile.txt and this time the resulting file contents are checked for garbage values. If the file contains garbage values, the key which produced this output would not have used for encrypting. This process is done for all the keys and in the end, only single key “box” does not produce the garbage while decrypted. This is the key used for encrypting.

Key used for encrypting: “box”

The message encrypted is: “A four-pound rock that left Mars 16 million years ago may hold the clues to ancient life on the planet. The rock is one of 13 Mars meteorites found on Earth. These rocks may contain clues to the ancient history of planet Mars, believed to have been a warmer, wetter place over 3 billion years ago. NASA's planetary scientists tell us why they think the Mars rock contains evidence of ancient life.”

Snapshot from command line showing same:

Initially probable keys are printed and in the end actual key used for encrypting and message are printed.



**Goal 2:**

The python script (goal2.py) for goal is present in Goal2 folder. The script is executed by “python goal2.py”. Two files fingerprint.txt and file.txt are generated. file.txt contains textual form of certificate and finger.txt contains fingerprint of CA certificate

**A) A copy of your certificate in textual form (using -text)**

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 4158 (0x103e)

Signature Algorithm: sha256WithRSAEncryption

Issuer: C=US, ST=NC, L=Raleigh, O=NCSU, OU=CSC, CN=574/emailAddress=harfoush@cs.ncsu.edu

Validity

Not Before: Nov 28 23:06:49 2015 GMT

Not After : Nov 27 23:06:49 2016 GMT

Subject: C=US, ST=North Carolina, L=Raleigh, O=NCSU, OU=CSC, CN=Mani Kumar/emailAddress=pghanta@ncsu.edu

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (1024 bit)

Modulus:

00:cb:03:d1:db:74:99:a5:1f:97:c8:16:ee:90:65:

72:2d:f6:e2:9c:c7:d9:c3:5d:cb:83:d0:51:6b:2e:

3b:4f:32:bc:64:c1:58:35:04:b1:3d:01:ac:2c:8f:

b9:47:e1:35:02:62:58:0d:58:6b:83:4d:89:e4:0f:

18:0f:7d:09:0d:7e:8e:ae:da:3c:63:0e:57:9f:40:

af:08:68:8d:4a:81:76:7c:d7:c5:3e:ba:c1:3d:73:

98:b8:77:ef:d6:4d:62:12:60:cd:c5:05:ce:58:8f:

e5:95:d4:88:25:93:60:5f:d1:32:1e:5f:32:2f:0d:

ff:06:4f:14:c1:1d:5d:e9:5f

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

46:18:24:DB:FB:92:24:AD:B2:79:1B:26:11:2A:AD:51:1B:87:50:95

X509v3 Authority Key Identifier:

keyid:AE:85:18:73:51:F3:9C:38:22:0B:55:8C:52:2F:C2:95:06:F9:9D:EF

Signature Algorithm: sha256WithRSAEncryption

0f:9c:f9:d4:53:9e:4e:b6:f7:7a:b2:12:22:45:72:24:a7:8e:

06:48:76:8c:e5:e1:12:fb:36:1d:98:77:55:f9:85:2e:7d:e9:

c7:17:67:03:9e:a9:58:ad:00:da:e4:05:20:4c:d7:c5:0d:29:

33:ed:91:40:2d:be:44:9d:e3:51:4e:f3:bc:e5:f1:9c:74:e9:

0c:bd:54:c0:76:38:c4:fa:b1:0a:a8:1b:23:ec:31:3b:9d:ed:

e6:15:e0:34:a3:8e:68:5d:95:3d:78:a6:63:e5:27:66:2c:3b:

d0:f6:ed:dd:d1:bd:e2:3f:af:05:b2:26:64:d9:37:4e:29:0d:

4e:70

-----BEGIN CERTIFICATE-----

MIIC+jCCAmOgAwIBAgICED4wDQYJKoZIhvcNAQELBQAwfDELMAkGA1UEBhMCVVMx

CzAJBgNVBAgMAk5DMRAwDgYDVQQHDAdSYWxlaWdoMQ0wCwYDVQQKDAROQ1NVMQww

CgYDVQQLDANDU0MxDDAKBgNVBAMMAzU3NDEjMCEGCSqGSIb3DQEJARYUaGFyZm91

c2hAY3MubmNzdS5lZHUwHhcNMTUxMTI4MjMwNjQ5WhcNMTYxMTI3MjMwNjQ5WjCB

izELMAkGA1UEBhMCVVMxFzAVBgNVBAgMDk5vcnRoIENhcm9saW5hMRAwDgYDVQQH

DAdSYWxlaWdoMQ0wCwYDVQQKDAROQ1NVMQwwCgYDVQQLDANDU0MxEzARBgNVBAMM

Ck1hbmkgS3VtYXIxHzAdBgkqhkiG9w0BCQEWEHBnaGFudGFAbmNzdS5lZHUwgZ8w

DQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMsD0dt0maUfl8gW7pBlci324pzH2cNd

y4PQUWsuO08yvGTBWDUEsT0BrCyPuUfhNQJiWA1Ya4NNieQPGA99CQ1+jq7aPGMO

V59ArwhojUqBdnzXxT66wT1zmLh379ZNYhJgzcUFzliP5ZXUiCWTYF/RMh5fMi8N

/wZPFMEdXelfAgMBAAGjezB5MAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9w

ZW5TU0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBRGGCTb+5IkrbJ5

GyYRKq1RG4dQlTAfBgNVHSMEGDAWgBSuhRhzUfOcOCILVYxSL8KVBvmd7zANBgkq

hkiG9w0BAQsFAAOBgQAPnPnUU55Otvd6shIiRXIkp44GSHaM5eES+zYdmHdV+YUu

fenHF2cDnqlYrQDa5AUgTNfFDSkz7ZFALb5EneNRTvO85fGcdOkMvVTAdjjE+rEK

qBsj7DE7ne3mFeA0o45oXZU9eKZj5SdmLDvQ9u3d0b3iP68FsiZk2TdOKQ1OcA==

-----END CERTIFICATE-----

B) Proof that you have the correct CA certificate (i.e., use OpenSSL to print out the fingerprint of the CA certificate and state it matches the fingerprint posted on the course web site. A web page will be created later in the semester with the students’ names, email addresses, and certificates)

SHA1 Fingerprint=EA:8A:F7:B7:4B:C7:E6:4B:59:E4:50:14:FA:88:D2:26:65:22:C4:23

Finger print in web SHA1 Fingerprint EA:8A:F7:B7:4B:C7:E6:4B:59:E4:50:14:FA:88:D2:26:65:22:C4:23

Fingerprint calculated from the root-ca.crt is same as given in webpage.

Below python script is used for saving certificate in text and to compute the finger print and save in fingerprint.txt

Python script used:

import sys

import os

from subprocess import call, Popen

import subprocess

v = Popen(['openssl', 'x509', '-in', '103E.pem', '-text'],stdout=subprocess.PIPE).communicate()

print v[0]

public = v[0]

pub\_file = open("text.txt", 'w+')

pub\_file.write(public)

pub\_file.close()

v = Popen(['openssl', 'x509', '-sha1', '-in', 'root-ca.crt', '-noout', '-fingerprint'],stdout=subprocess.PIPE).communicate()

print v[0]

public = v[0]

pub\_file = open("fingerprint.txt", 'w+')

pub\_file.write(public)

pub\_file.close()

**Goal 3:**

All the files needed for goal 3 are present in Goal3 folder.

Files needed to be present for the program to work:

“mykey.pem” – This keys contain our private key. This key is used for signing while sending and to decrypt session key while receiving.

“goal3.py” – Python script containing the functionalities.

“root-ca.crt” – To verify the certificate downloaded from the repository with the root-ca.crt.

**Explanation sending mail sub routine**

**def send\_mail(inmsg, emailID):**

Initially (approx. first 30 lines of the code) the certificate of recipient is checked in local database. If the certificate of recipient exists in local database, certificate need not be verified and can be used. If the certificate is not present in local database, the certificate is downloaded from repository and verified. If certificate verification fails, certificate is not inserted in local data base and message sending fails.

If the certificate verification passes, then 32 byte random session key is generated, stored in file skey.txt and encrypted with the public key of the recipient using the certificate. Command used is below

status = call(['openssl', 'rsautl', '-encrypt', '-inkey', filepath, '-certin', '-in', 'skey.txt', '-out', 'encskey.txt'])

The encrypted session key is stored in encskey.txt

Next, message is encrypted with session key using below command and stored the output in encinmsg.txt

status = call(['openssl', 'enc', '-aes-256-cbc', '-base64', '-in', 'inmsg.txt', '-k', key, '-out', 'encinmsg.txt'])

Next, encrypted session key + blank line + encrypted message with session key is digested using sha1 using below commands

concat\_msg = encskey + '\n' + blankmsg + encinmsg

msg\_to\_hash=open("message\_to\_hash.txt",'w+')

msg\_to\_hash.write(concat\_msg)

msg\_to\_hash.close()

status = call(['openssl', 'dgst', '-sha1', '-out', 'sha.txt', 'message\_to\_hash.txt'])

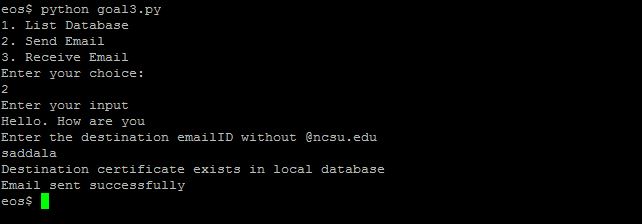
Next, message hash is taken and signed using our private key using below command and output stored in sign.txt

status = call(['openssl', 'rsautl', '-sign', '-inkey', 'mykey.pem', '-keyform', 'PEM', '-in', 'hash.txt', '-out', 'sign.txt'])

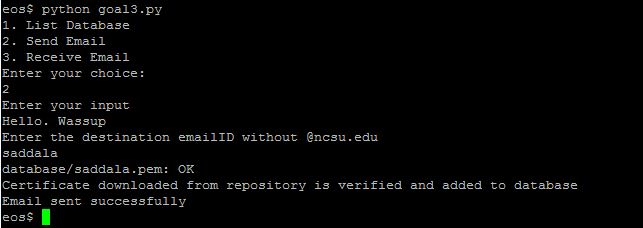
Next, all the messages, encskey + '\n' + blankmsg + encinmsg + blankmsg + sign are written in the output file along with Email header and -----BEGIN CSC 574 MESSAGE-----, -----END CSC 574 MESSAGE-----

Final message is stored as email\_msg.txt. Screenshots of working are given below

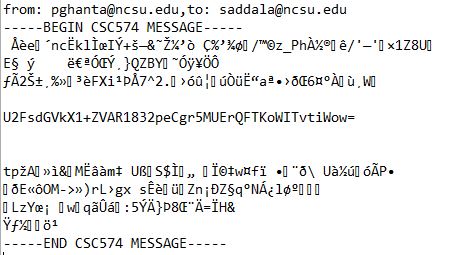
Screenshot of sending (Destination certificate in local database)



Screenshot of sending (Destination certificate downloaded from repository)



Contents of message for second case screenshot



**Explanation receiving mail sub routine**

**def receive\_mail(recv\_file):**

Initially, the received message is split using “\n\n” delimiters to store the message in 3 parts. First containing the Email header and session key encrypted. Second message contains the message encrypted with session key. Third containing the sign and END header.

Again the messages are split to find the sender email and stored in fromEmail, encrypted sign is stored in encRecvsign.txt, encypted message is stored in encRcvmsg.txt, encrypted session key stored in encRcvSkey.txt.

Certificate of the sender is downloaded from certificate repository if not present in local repository and verified.

First received sign is decrypted using sender’s public key using below command

status = call(['openssl', 'rsautl', '-inkey', 'senderPubkey.pem', '-pubin', '-in', 'encRecvsign.txt', '-out', 'decRecvsign.txt'])

Next, SHA1 digest is computed on encRcvSkey + blankline + encRcvmsg using below command

rcvd\_concat\_msg = encRcvSkey + "\n" + "\n" + encRcvmsg

rcvd\_msg\_to\_hash=open("rcvd\_message\_to\_hash.txt",'w+')

rcvd\_msg\_to\_hash.write(rcvd\_concat\_msg)

rcvd\_msg\_to\_hash.close()

status = call(['openssl', 'dgst', '-sha1', '-out', 'rcvd\_sha.txt', 'rcvd\_message\_to\_hash.txt'])

Now both digests are compared to check the signature.

if v[1] == decRecvsign:

print "Signature in the mail is verified"

else:

print "Signature verification failed. Message rejected!"

return

Next, encrypted session key is decrypted using own private key using below command

status = call(['openssl','rsautl','-decrypt','-in','encRcvSkey.txt','-inkey','mykey.pem', '-out', 'recv\_skey.txt'])

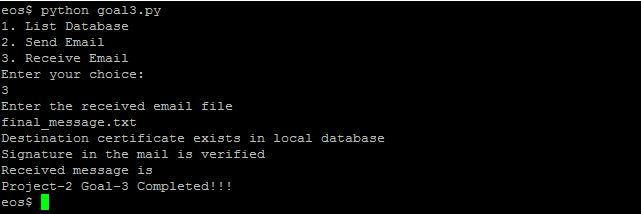
At the end, encrypted received message is decrypted using session key using below command

status = call(['openssl', 'enc', '-aes-256-cbc', '-base64', '-d', '-in', 'encRcvmsg.txt', '-k', rcvd\_skey, '-out', 'rcvd\_msg.txt'])

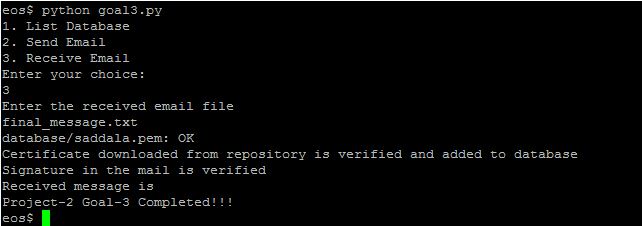
Also the received message is printed on the command prompt.

Screenshots of the received email is shown below

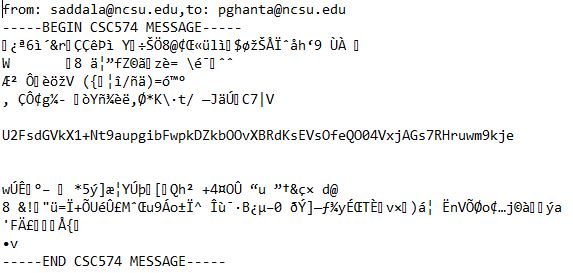
Screenshots of the received mail (Sender certificate in local database):



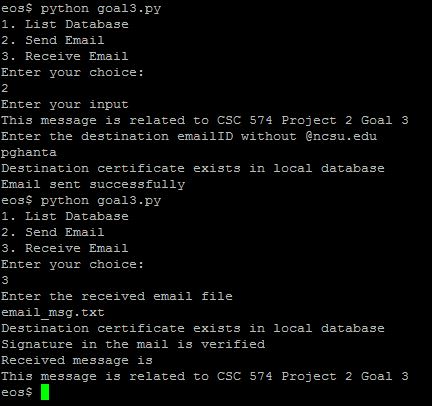
Screenshots of the received mail (Sender certificate downloaded from repository):



Contents of received Final message:



Also, including sending email to self and decrypting the same email as receiver:



List database shows all the certificate present in the database:

Screenshot is shown below:

