Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Network Security
Course Number:	COE817
Semester/Year (e.g.F2016)	W2022

Instructor:	Khalid Abdel Hafeez

Assignment/Lab Number:	03
Assignment/Lab Title:	Lab 3

Submission Date:	March 8 2022
Due Date:	March 11 2022

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
VISWANATHAN	SRIVATSAN	500895214	02	S.V

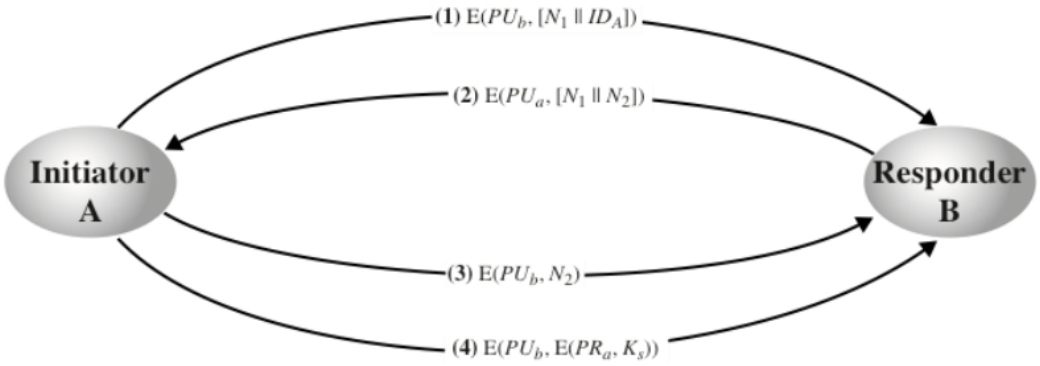
*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

1 Objectives

The objective of this lab is to create a secure chat system by using RSA and DES algorithms in Python or Java.

2 Key Distribution

2.1 Public key based distribution



Step 1: A uses B's public key to encrypt a message containing A's identifier and a nonce which is used to prove authenticity. The message is sent to B.

Step 2: B uses A's public key to encrypt a message containing A's nonce and B's nonce (B generates it's nonce). The message is sent to A.

Step 3: A uses B's public key to encrypt a message containing B's nonce to confirm B is corresponding with A. The message is sent to B.

Step 4: A uses A's private key and encrypts a secret key with it. A then encrypts the result of that with B's public key. The message is sent to B.

Step 5: B uses B's private key to decrypt the message from A. B then uses A's public key to decrypt the result which results in recovering the secret key from A.

2.2 Client

```
Message 1:  b'55204381INITIATOR A'
Encrypted message 1:  b'_\xe3\xaf\xf6\x89\x80HXB,-
mF\x10\x9b\xa1N\x17,\xb7,\xd1~Y\xfa\x0e\x07\xb60W\xc6\xc8\x0b\xa6g\xfa\xaa\x17\n29\x16\xcf\x98RD\x99q\x7f\t\
xf0\xf5_\xd8\x87\x12i*\xc9\x0b\xc2|U%O1\x08\xa4\x1a\xe3bg\xc9[\t\n}T\xcd+\xd0\xe4\x17&n\xa4\x03E!\x10\xfeN\xf7
\xaa\xe5\x81\\\xb8\xa3|\xd4\xc0\x00\xf7\xe8\xad\x1fu\x80;"r\xe4\xfe\xf7\x9bH] \xd69\xd9\xb6L\xd1\xb5\xa1\x05'
Received encrypted message 2:
b"i\x98\xda\x8cos\xdd\x93\xa4\xc0\xdd\x07\x85\xf6\xdc+\xf7\x03R\x18\x87\t\xc6\x92\xeeG\xd4\x98\xaah\xc0\xdb\x
b7\xbc\xc0\xb437\x01G\xf9\xefv\xa0g\t\xdb\xf4\xe77c\xa2\xdc\x0e-zJW-
\xde\xd6'P\xe0\xdb\x1c\x97D>\x94\x80\xd4'Mal\x8b\x84\x99\xba\x92\xe7\xdc\x9d\xe0c\x94\xfa\x03\x08$\x1b7\xb3\
xda\xac\xf8D\xda\xd7\xa2\x86/"\xa2R\xa8\xf3g\xe2\xf84\x99\xf5\xcat'\xf0\xb3ZaK\xcaX#,\xc8\x92"
Received message 2:  b'5520438129243223'
Message 3:  b'29243223'
Encrypted message 3:
b'\xa9zB\xcc\xd9\xd2\x15\xef\xb9\r\xc5i\xf5\x8d\x83(\xc8x\xbcP\x99\xcf\xd0\xd6\x15\x82;+\x07I.\x96\xc4\x86\xdeV\x
ce\x1a\xe9o\xf6\xe4\x80O\xda\xd1\x00\xf6\xf4"n/\xaa\xf7\xa5}\xc5\x1d\x0e\xd1#\xacg\xb3\xe9S\xd1\x91\x16VO\xf5\
xd5\x19\xfcsV\xd1\x8d`Z\x88N\x83~k\xd4\xcc\x88\xe7\xc17\x99\x88\x91\xb8\nS7\xe4\xa6\xfb\xc8"7\x90\xb1\x8f
\x00Y\xe2\xf1\xdc\xb9\xd0\x07\xd0[\xf3\x05\x90\xf2\xdb\xa8\xfe\x95\xc8'
Session Key:  b'T\xcc\xc7\xe2\xeb\xaf\x7fu'
Encrypted message 4:
b"z\xf5\x14O7\x95\x9b\xf0]Fv\xd5\x95\xfb\xe4\nsa4e\xad'9E\x0f\x81\x7f\xe5\xef\xc6\xb2\x18\x90\xa0\xcf\xb4+._\xf0\
xfa\xdf\xddX\x1a\xa1\xe0\xa9L\xbd\xc87\xd1\xa8;z\x87\xc6\x85\x0f\xc4dX\x9aX\x19\xe5\x9fb-
\x03\x17\x01_\x0b!r\x17\xe9\xb92\xa6\xb0\xe7\xffF\xff\x91\xaeJ\xb23\x16\xcb\x8c\x06\xb6\x0e`\x0fyR\xd8\xa3\xb9\x
ea\x05\xbf\xcf\xc9\xe6\xa0/\xf3\x159\r\xdbs\xf5\x8c\x92\t\x85\xd9,J\x90"
```

2.3 Server

```
Received encrypted message 1:  b'_\xe3\xaf\xf6\x89\x80HXB,-
mF\x10\x9b\xa1N\x17,\xb7,\xd1~Y\xfa\x0e\x07\xb60W\xc6\xc8\x0b\xa6g\xfa\xaa\x17\n29\x16\xcf\x98RD\x99q\x7f\t\xf0\x
f5_\xd8\x87\x12i*\xc9\x0b\xc2|U%O1\x08\xa4\x1a\xe3bg\xc9[\t\n}T\xcd+\xd0\xe4\x17&n\xa4\x03E!\x10\xfeN\xf7\xaa\xe5\
x81\\\xb8\xa3|\xd4\xc0\x00\xf7\xe8\xad\x1fu\x80;"r\xe4\xfe\xf7\x9bH] \xd69\xd9\xb6L\xd1\xb5\xa1\x05'
Received message 1:  b'55204381INITIATOR A'
Message 2:  5520438129243223
Encrypted message 2:
b"i\x98\xda\x8cos\xdd\x93\xa4\xc0\xdd\x07\x85\xf6\xdc+\xf7\x03R\x18\x87\t\xc6\x92\xeeG\xd4\x98\xaah\xc0\xdb\xb7\x
bc\xc0\xb437\x01G\xf9\xefv\xa0g\t\xdb\xf4\xe77c\xa2\xdc\x0e-zJW-
\xde\xd6'P\xe0\xdb\x1c\x97D>\x94\x80\xd4'Mal\x8b\x84\x99\xba\x92\xe7\xdc\x9d\xe0c\x94\xfa\x03\x08$\x1b7\xb3\xda\
xac\xf8D\xda\xd7\xa2\x86/"\xa2R\xa8\xf3g\xe2\xf84\x99\xf5\xcat'\xf0\xb3ZaK\xcaX#,\xc8\x92"
Received encrypted message 3:
b'\xa9zB\xcc\xd9\xd2\x15\xef\xb9\r\xc5i\xf5\x8d\x83(\xc8x\xbcP\x99\xcf\xd0\xd6\x15\x82;+\x07I.\x96\xc4\x86\xdeV\xce\x
1a\xe9o\xf6\xe4\x80O\xda\xd1\x00\xf6\xf4"n/\xaa\xf7\xa5}\xc5\x1d\x0e\xd1#\xacg\xb3\xe9S\xd1\x91\x16VO\xf5\xd5\x19\
xfcsV\xd1\x8d`Z\x88N\x83~k\xd4\xcc\x88\xe7\xc17\x99\x88\x91\xb8\nS7\xe4\xa6\xfb\xc8"7\x90\xb1\x8f
\x00Y\xe2\xf1\xdc\xb9\xd0\x07\xd0[\xf3\x05\x90\xf2\xdb\xa8\xfe\x95\xc8'
Received message 3:  b'29243223'
Received encrypted message 4:
b"z\xf5\x14O7\x95\x9b\xf0]Fv\xd5\x95\xfb\xe4\nsa4e\xad'9E\x0f\x81\x7f\xe5\xef\xc6\xb2\x18\x90\xa0\xcf\xb4+._\xf0\xfa\
xdf\xddX\x1a\xa1\xe0\xa9L\xbd\xc87\xd1\xa8;z\x87\xc6\x85\x0f\xc4dX\x9aX\x19\xe5\x9fb-
\x03\x17\x01_\x0b!r\x17\xe9\xb92\xa6\xb0\xe7\xffF\xff\x91\xaeJ\xb23\x16\xcb\x8c\x06\xb6\x0e`\x0fyR\xd8\xa3\xb9\xea\x
05\xbf\xcf\xc9\xe6\xa0/\xf3\x159\r\xdbs\xf5\x8c\x92\t\x85\xd9,J\x90"
Received session key:  b'T\xcc\xc7\xe2\xeb\xaf\x7fu'
```

3 Chat Message

3.1 Client

```
b'Messaging Application:'
Message:  hello
Encrypted Message:  b'g\x03\x82\x11.\xad5ui\x7f\xa7\nY'
Message:  this is lab 3
Encrypted Message:  b'S\x1c\x0f\xd1.kp\x14U\t\x13Z\xd4\xcf\x0fj\xf3\x96A\x88X'
```

3.2 Server

```
Messaging Application:
Received encrypted message:  b'g\x03\x82\x11.\xad5ui\x7f\xa7\nY'
Client sent:  b'hello'
Received encrypted message:  b'S\x1c\x0f\xd1.kp\x14U\t\x13Z\xd4\xcf\x0fj\xf3\x96A\x88X'
Client sent:  b'this is lab 3'
```

4 Conclusion

A socket program with authenticated key distribution between the client and server was created. The chat messages for the client and server and encrypted through the DES algorithm. To stop combat replay attacks, you could create a unique password for each message that the client and user could use. If an attacker tries to replay an attack, the password will have become null already (expired) so they can't replay the attacks.

5 Appendix

5.1 Client

```
import socket                    # Import socket module
from Crypto.PublicKey import RSA
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from Crypto.Cipher import PKCS1_OAEP
from PIL import Image
import random
import ast

def key_encrypt(message):
    rsa_public_key = RSA.importKey(B_public_key)
    rsa_public_key = PKCS1_OAEP.new(rsa_public_key)
    encrypted = rsa_public_key.encrypt(message)
    return encrypted

def key_encrypt2(message):
    rsa_private_key = RSA.importKey(private_key)
    rsa_private_key = PKCS1_OAEP.new(rsa_private_key)
    encrypted = rsa_private_key.encrypt(message)
    return encrypted

def key_decrypt(encrypted):
    rsa_private_key = RSA.importKey(private_key)
    rsa_private_key = PKCS1_OAEP.new(rsa_private_key)
    decrypted = rsa_private_key.decrypt(encrypted)
    return decrypted

def nonce_creator():
    length = 8
    nonce = ''
    for i in range(length):
        nonce = nonce + str(random.randint(0,9))
    return nonce
```

```
def generate_key():
    key = get_random_bytes(8)
    return key

def encrypt(Plaintext_pad, key):
    nonce = get_random_bytes(8)
    cipher = DES.new(key, DES.MODE_OFB, nonce)
    encrypted_message = cipher.encrypt(Plaintext_pad)
    return nonce + encrypted_message

def decrypt(ciphertext, key):
    nonce = ciphertext[:8]
    ciphertext = ciphertext[8:]
    cipher = DES.new(key, DES.MODE_OFB, nonce)
    decrypted_message = cipher.decrypt(ciphertext)
    return nonce + decrypted_message

s = socket.socket()             # Create a socket object
port = 60000                    # Reserve a port for your service.

s.connect (('127.0.0.1', port))

key = RSA.generate(1024)
public_key = key.publickey().exportKey()
private_key = key.exportKey()

s.send(public_key)
B_public_key = s.recv(1024)

# Part 1:
nonce = nonce_creator()
message1 = nonce + 'INITIATOR A'
message1 = str.encode(message1)
encrypted_message1 = key_encrypt(message1)
s.send(encrypted_message1)
print("Message 1: ", message1)
print("Encrypted message 1: ", encrypted_message1)

# Part 2:
encrypted_message2 = s.recv(1024)
message2 = key_decrypt(ast.literal_eval(str(encrypted_message2)))
nonce2 = message2[8:]
print("Received encrypted message 2: ", encrypted_message2)
print("Received message 2: ", message2)

# Part 3:
```

```
message3 = nonce2
encrypted_message3 = key_encrypt(message3)
s.send(encrypted_message3)
print("Message 3: ", message3)
print("Encrypted message 3: ", encrypted_message3)

# Part 4:
session_key  = generate_key()
message4 = key_encrypt2(session_key)
encrypted_message4 = key_decrypt(message4)
encrypted_message4 = key_encrypt(encrypted_message4)
s.send((encrypted_message4))
print("Session Key: ", session_key)
print("Encrypted message 4: ", encrypted_message4)

# DES Message:
auth_message = s.recv(1024)
print(auth_message)
send = True

 file  = open("lelouch.jpg", "rb")
image = file.read()
s.send(image)

while send:
    message = input("Please enter a message you want to send to the server: \n")
    encrypted_message = encrypt(message.encode("utf-8"), session_key)
    print("Message: ", message)
    print("Encrypted Message: ", encrypted_message)
    s.send(encrypted_message)
    if  message == '0':
        send = False
```

5.2 Server

```
import socket                   # Import socket module
from Crypto.PublicKey import RSA
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from Crypto.Cipher import PKCS1_OAEP
from PIL import Image
import random
import ast

port = 60000                    # Reserve a port for your service.
```

```
s = socket.socket()                 # Create a socket object
host = socket.gethostname()         # Get local machine name
print(host)
s.bind (('127.0.0.1',  port))       # Bind to the port
s. listen (5)                       # Now wait for client connection.

def key_encrypt(message):
    rsa_public_key = RSA.importKey(A_public_key)
    rsa_public_key = PKCS1_OAEP.new(rsa_public_key)
    encrypted = rsa_public_key.encrypt(message)
    return encrypted

def key_encrypt2(message):
    rsa_private_key = RSA.importKey(private_key)
    rsa_private_key = PKCS1_OAEP.new(rsa_private_key)
    encrypted = rsa_private_key.encrypt(message)
    return encrypted

def key_decrypt(encrypted):
    rsa_private_key = RSA.importKey(private_key)
    rsa_private_key = PKCS1_OAEP.new(rsa_private_key)
    decrypted = rsa_private_key.decrypt(encrypted)
    return decrypted

def nonce_creator():
    length = 8
    nonce = ''
    for i in range(length):
        nonce = nonce + str(random.randint(0,9))
    return nonce

def encrypt(Plaintext_pad, key):
    nonce = get_random_bytes(8)
    cipher = DES.new(key.encode(), DES.MODE_OFB, nonce)
    encrypted_message = cipher.encrypt(Plaintext_pad.encode())
    return nonce + encrypted_message

def decrypt(ciphertext, key):
    nonce = ciphertext[:8]
    ciphertext = ciphertext[8:]
    cipher = DES.new(key, DES.MODE_OFB, nonce)
    decrypted_message = cipher.decrypt(ciphertext)
    return decrypted_message

key = RSA.generate(1024)
public_key = key.publickey().exportKey()
```

```
private_key = key.exportKey()

conn, addr = s.accept()       # Establish connection with client.

conn.send(public_key)
A_public_key = conn.recv(1024)

# Part 1:
encrypted_message1 = conn.recv(1024)
message1 = key_decrypt(ast.literal_eval(str(encrypted_message1)))
nonce1 = message1.decode("utf-8")[:8]
print("Received encrypted message 1: ", encrypted_message1)
print("Received message 1: ", message1)

# Part 2:
nonce2 = nonce_creator()
message2 = nonce1 + nonce2
encrypted_message2 = key_encrypt(message2.encode("utf-8"))
conn.send(encrypted_message2)
print("Message 2: ", message2)
print("Encrypted message 2: ", encrypted_message2)

# Part 3:
encrypted_message3 = conn.recv(1024)
message3 = key_decrypt(ast.literal_eval(str(encrypted_message3)))
print("Received encrypted message 3: ", encrypted_message3)
print("Received message 3: ", message3)

# Part 4:
encrypted_message4 = conn.recv(1024)
message4 = key_decrypt(ast.literal_eval(str(encrypted_message4)))
session_key = key_encrypt2(message4)
session_key = key_decrypt(session_key)
print("Received encrypted message 4: ", encrypted_message4)
print("Received session key: ", session_key)



# DES Messaging:
receive = True
conn.send(bytes("Messaging Application:", "utf-8"))

file = open("img.jpg", "wb")
image = conn.recv(170000)
file.write(image)
img = Image.open("img.jpg")
```

```
img.show()

print("Messaging Application:")
while receive:
    encrypted_message = conn.recv(1024)
    print("Received encrypted message: ", encrypted_message)
    message = decrypt(encrypted_message, session_key)
    print("Client sent: ", message)

    if message == '0':
        receive = False
```