The following diagram shows key size (in Kilo Bits) on the X axis and time (in microsecond) in Y axis for encrypting & Decrypting 256 Byte value with RSA Cryptosystem. In following cases, encryption is done with public key and decryption is done with private key.

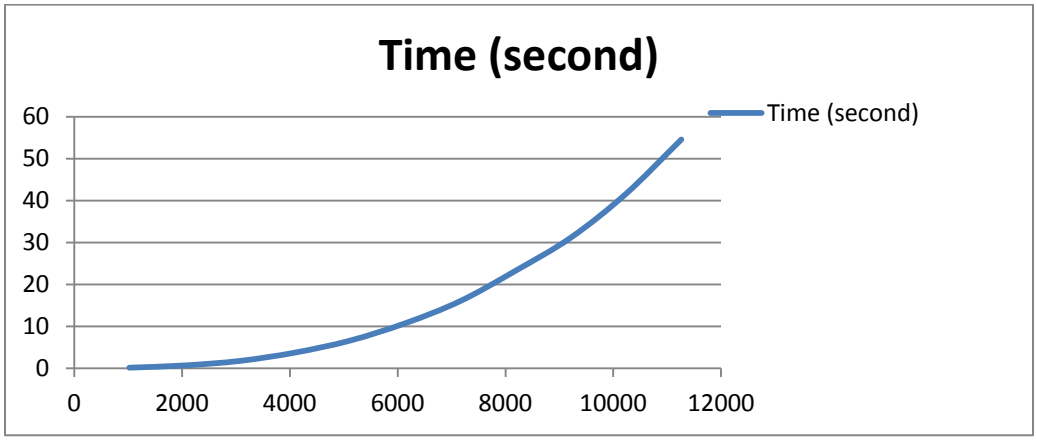

Fig 1: Key size (X-Axis in Bits) vs encryption-decryption time (Y-Axis in sec) of a single run

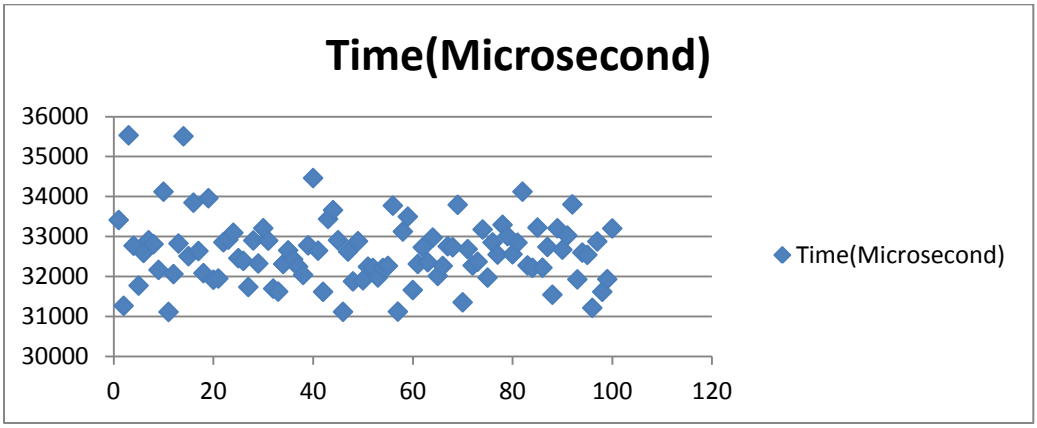

Fig 2: Encryption & Decryption time of 100 random messages with same key size (1024 bit). Y axis represent required time & X axis represent random messages

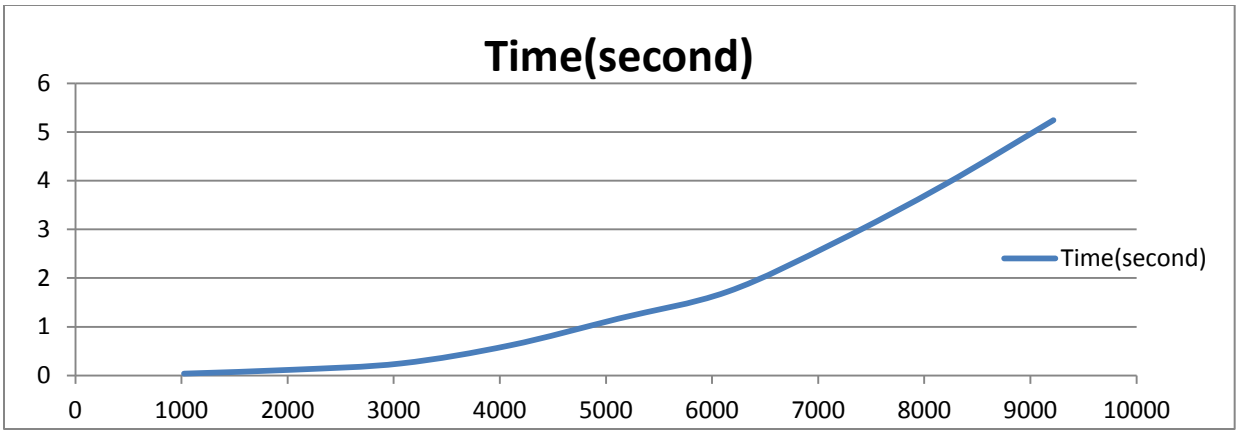

Fig 3: Average Encryption-Decryption time of 100 random Messages over different key size. X axis represent key size in bits and Y axis represent time in second.

Observation:

1. The time required to encrypt and decrypt a string with different keysizes vary significantly in the experiment.
2. I have implementedhw4 in python language with Pycrypto Library [1] (default with python installation) which does not allow RSA encryption with private key and RSA decryption with public key. Following is an excerpt from [2]

   "You can encrypt something with the private key as the private key contains the information required to make the public key, but it would be unusual to do so, as normally the person encrypting the data does* not *have the private key"

References:

[1] https://pypi.python.org/pypi/pycrypto

[2] http://stackoverflow.com/questions/9893080/pycrypto-decrypt-only-with-public-key-in-file-no-privatepublic-key

Appendix 1: Python code for HW4

```
from  Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import datetime, time
import random

def generate_RSA(keysize=1024):
    rsa = RSA.generate(keysize)
    publicKey = rsa.publickey().exportKey()
    privateKey = rsa.exportKey()
    return publicKey, privateKey

def createMessage(size=256):
```

```
    return ''.join(chr(random.randint(97,122)) for i in range(0,255))


def encryptNDecryptLongMessage (pub_key,priv_key, msg=None):

    if msg == None:
        msg = createMessage()

    chunks = msg.__len__()
    chunk_size = 50
    splitedStr = [ msg[i:i+chunk_size] for i in range(0, chunks, chunk_size) ]

    stime = datetime.datetime.now()
    for msg in splitedStr :
        encrypted_msg = encrypt(pub_key,msg)
        decrypted_msg = decrypt(priv_key,encrypted_msg)
        #print encrypted_msg, decrypted_msg
    end_time = datetime.datetime.now()
    return stime, end_time

def encrypt(key, msg):

    publicKey = RSA.importKey(key)
    cipher = PKCS1_OAEP.new(publicKey)
    cipher_text = cipher.encrypt(msg)
    return cipher_text

def decrypt (key, encypted_msg=None):
    privKey = RSA.importKey(key)
    cipher = PKCS1_OAEP.new(privKey)
    message = cipher.decrypt(encypted_msg)
    return message

def hw4():
    keysize = [ i*1024 for i in range(1,20) ]

    for size in keysize:
        (pub_key, priv_key) = generate_RSA(size)
        (s1_time,end1_time) = encryptNDecryptLongMessage(pub_key,priv_key)
        #(s2_time,end2_time) = encryptNDecryptLongMessage(priv_key,pub_key)
        print size, end1_time - s1_time

def timestamp(date):
    return time.mktime(date.timetuple())

def diffKeySize100MsgTest():
    keysize = [ i*1024 for i in range(1,10) ]
    for size in keysize:
      avg=0
      for i in range(1,101):
        (pub_key, priv_key) = generate_RSA(size)
        (s1_time,end1_time) = encryptNDecryptLongMessage(pub_key,priv_key)
        avg = (avg * (i -1) + (timestamp (end1_time) - timestamp (s1_time) ))/i
        #(s2_time,end2_time) = encryptNDecryptLongMessage(priv_key,pub_key)
      print size, avg


def diffKeySingleMessage():
   keysize = [ 1024 for i in range(1,101) ]
   for size in keysize:
        (pub_key, priv_key) = generate_RSA(size)
        (s1_time,end1_time) = encryptNDecryptLongMessage(pub_key,priv_key)
        #req_time =  (timestamp (end1_time) - timestamp (s1_time) )
        #(s2_time,end2_time) = encryptNDecryptLongMessage(priv_key,pub_key)
        print size, end1_time - s1_time

#diffKeySingleMessage()
diffKeySize100MsgTest()
```