**CSE3501-INFORMATION SECURITY ANALYSIS AND AUDIT**

**Lab: L9+L10**

**Lab Assignment-5**

**NAME:  Alokam Nikhitha**

**REG NO: 19BCE2555**

Aim

To encrypt a message with RSA and to calculate the time taken by the algorithm to decrypt the same message compared to a brute force attack to guess the key.

**TIMING ATTACKS**

A timing attack is a rather sophisticated way to circumvent the security mechanisms of an application. In a timing attack, the attacker gains information that is indirectly leaked by the application. This information is then used for malicious purposes, such as guessing the password of a user.

It is a security hack that allows an attacker to notice flaws in a computer’s security by seeing how long it takes for the system to respond to different inputs.

Timing characteristics will vary depending upon on the [encryption](https://searchsecurity.techtarget.com/definition/encryption) [key](https://searchsecurity.techtarget.com/definition/key) because different systems take slightly different amounts of time to process different inputs. Variables include performance optimizations, branching and conditional statements, processor instructions, [RAM](https://searchstorage.techtarget.com/definition/RAM-random-access-memory) and [cache](https://searchstorage.techtarget.com/definition/cache) hits. A timing attack looks at how long it takes a system to do something and uses [statistical analysis](https://whatis.techtarget.com/definition/statistical-analysis) to find the right decryption key and gain access.

The canonical example of a timing attack was designed by cryptographer Paul Kocher. He was able to expose the private decryption keys used by [RSA](https://searchsecurity.techtarget.com/definition/RSA) encryption without breaking RSA. Timing attacks are also used to target devices such as smartcards and web servers that use [OpenSSL](https://whatis.techtarget.com/definition/OpenSSL). Web servers were believed to be less vulnerable to timing attacks because network conditions could mask differences in timing; recent research has challenged that assumption.

Different systems process different inputs in different periods of time, depending on different characteristics such as performance enhancements, branching, conditional statements, CPU instructions. The timing attack examines how long it takes for the system to complete a task, and employs statistical analysis to locate the correct description key.

**CODE** :

**RSA ENCRYPTION AND DECRYPTION**

import random

import time

def gcd(a, b):

    while b !=0:

        a, b = b, a % b

    return a

def multiplicative\_inverse(e, phi):

    d =0

    x1 =0

    x2 =1

    y1 =1

    temp\_phi = phi

    while e > 0:

        temp1 = temp\_phi//e

        temp2 = temp\_phi -temp1 \* e

        temp\_phi = e

        e=temp2

        x = x2 -temp1 \* x1

        y = d -temp1 \* y1

        x2 = x1

        x1 = x

        d = y1

        y1 = y

        if temp\_phi ==1:

            return d + phi

def is\_prime(num):

    if num ==2:

        return True

    if num <2 or num % 2==0:

        return False

    for n in range(3, int(num\*\*0.5)+2,2):

        if num % n ==0:

            return False

        return True

def generate\_key\_pair(p, q):

    if not(is\_prime(p)and is\_prime(q)):

        raise ValueError('Both numbers must be prime.')

    elif p == q:

        raise ValueError('p and q cannot be equal')

    n = p \* q

    phi = (p-1) \* (q-1)

    e = random.randrange(1, phi)

    g = gcd(e, phi)

    while g !=1:

        e = random.randrange(1, phi)

        g = gcd(e, phi)

    d = multiplicative\_inverse(e, phi)

    print("e: ", e)

    print("n: ", n)

    print("d: ", d)

    return ((e, n), (d, n))

def encrypt(pk, plaintext):

    key, n = pk

    cipher = [pow(ord(char), key, n)for char in plaintext]

    return cipher

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

    print("RSA-Encryption")

    p = int(input(" -Enter a prime number (17, 13,23, etc): "))

    q = int(input(" -Enter another prime number (Notone you enteredabove): "))

    print(" -Generating your public / private key-pairsnow . . .")

    public, private = generate\_key\_pair(p, q)

    print(" -Your public key is ", public," andyour private key is ",private)

    message ="abcdefghijklman"

    print(" ")

    encrypted\_msg = encrypt(public, message)

    print(" -Your encrypted message is: ",''.join(map(lambda x: str(x),encrypted\_msg)))

    start = time.time()

def decrypt(pk, ciphertext):

    key, n = pk

    aux = [str(pow(char, key, n))for char in ciphertext]

    plain = [chr(int(char2))for char2 in aux]

    return''.join(plain)

print(" -Decrypting message with private key ", private," . . .")

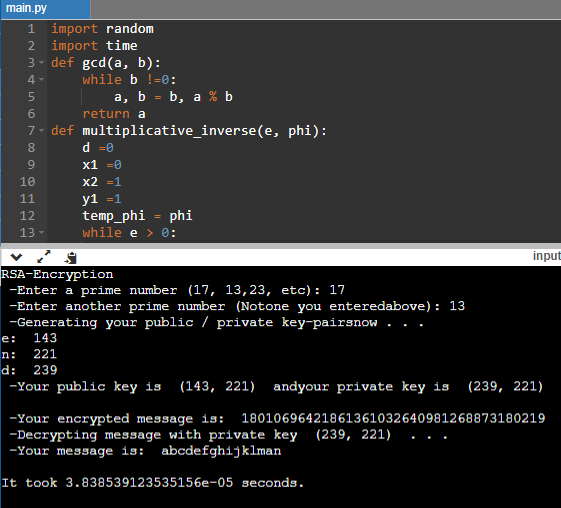
print(" -Your message is: ", decrypt(private, encrypted\_msg))

print(" ")

print('It took', time.time()-start,'seconds.')

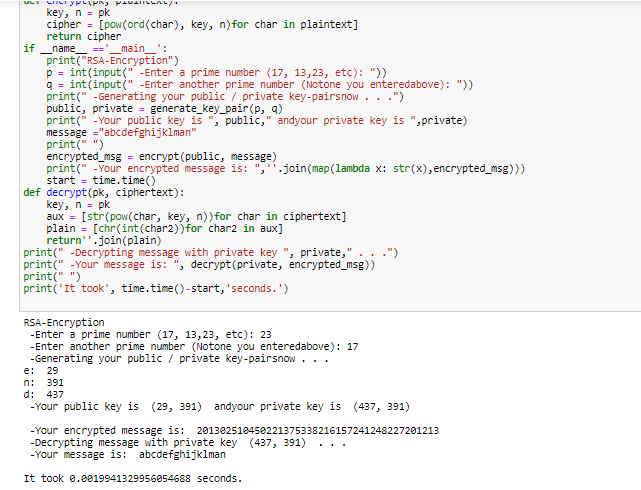
**OUTPUT**

Onlinegd.com (Time Taken = 0.000038385391235156 seconds.)



**Jupyter(Time  Taken=** 0.0019941329956054688

**seconds.)**



**BRUTE FORCE METHOD FOR DECRYPTION**

**CODE**

import time

start = time.time()

def egcd(a, b):

    if a==0:

        return(b,0,1)

    else:

        g, y, x = egcd(b %a,a)

        return (g, x -(b// a) \* y, y)

def modinv(a, m):

    g, x, y = egcd(a, m)

    if g !=1:

        raise Exception('modular inverse does notexist')

    else:

        return x % m

def factor(n):

    for i in range(3, n):

        if n%i ==0:

            return i

e =2815

n =4661

p = factor(n)

q =n//p

phi\_n = (p-1) \* (q-1)

d\_crack = modinv(e, phi\_n)

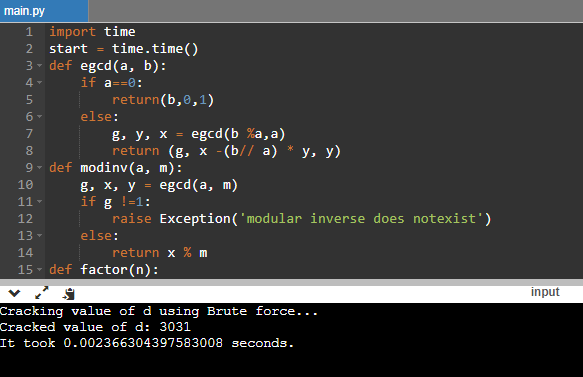
print("Cracking value of d using Brute force...")

print('Cracked value of d:', d\_crack)

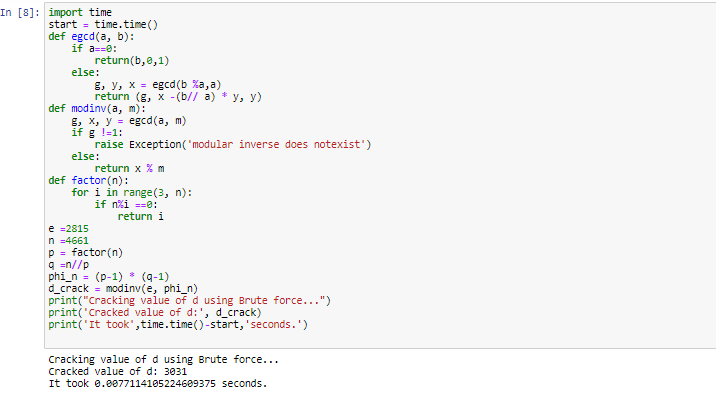
print('It took',time.time()-start,'seconds.')

**OUTPUT**

Onlinegdb.com (Time Taken = 0.002366304397583008seconds)



Jupyter ( Time Taken= 0.0077114105224609375seconds.)



 Results:

* **Time taken for RSA ENCRYPTION AND DECRYPTION**
* Onlinegd.com (Time Taken = 0.000038385391235156 seconds.)
* **Jupyter(Time  Taken=** 0.0019941329956054688
  + **seconds.)**
* **Time taken for BRUTE FORCE METHOD FOR DECRYPTION**
* Onlinegdb.com (Time Taken = 0.002366304397583008seconds)
* Jupyter ( Time Taken= 0.0077114105224609375seconds.)