INFORMATION SECURITY ANALYSIS AND AUDIT

NAME: Alokam Nikhitha

REG NO: 19BCE2555

TIMING ATTACKS

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.
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 = 0.0000383853912351 56 seconds.)

main.py

```
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:
```

input

```
RSA-Encryption
 -Enter a prime number (17, 13,23, etc): 17
 -Enter another prime number (Notone you enteredabove): 13
 -Generating your public / private key-pairsnow . . .
e:  143
n:  221
d:  239
 -Your public key is  (143, 221)  andyour private key is  (239, 221)

 -Your encrypted message is:  180106964218613610326409812688731802 19
 -Decrypting message with private key  (239, 221)  . . .
 -Your message is:  abcdefghijklman

It took 3.838539123535156e-05 seconds.
```

Jupyter(Time Taken= 0.0019941329956054688 seconds.)

```
    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.')
```

```
RSA-Encryption
 -Enter a prime number (17, 13,23, etc): 23
 -Enter another prime number (Notone you enteredabove): 17
 -Generating your public / private key-pairsnow . . .
e:  29
n:  391
d:  437
 -Your public key is  (29, 391)  andyour private key is  (437, 391)

 -Your encrypted message is:  201302510450221375338216157241248227201213
 -Decrypting message with private key  (437, 391)  . . .
 -Your message is:  abcdefghijklman

It took 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 = 0.0023663043975830 08seconds)

main.py

```
1  import time
2  start = time.time()
3  def egcd(a, b):
4      if a==0:
5          return(b,0,1)
6      else:
7          g, y, x = egcd(b %a,a)
8          return (g, x -(b// a) * y, y)
9  def modinv(a, m):
10     g, x, y = egcd(a, m)
11     if g !=1:
12         raise Exception('modular inverse does notexist')
13     else:
14         return x % m
15 def factor(n):
```

input

```
Cracking value of d using Brute force...
Cracked value of d: 3031
It took 0.002366304397583008 seconds.
```

Jupyter (Time Taken= 0.007711410522460937 5seconds.)

In [8]:
```
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.')
```

```
Cracking value of d using Brute force...
Cracked value of d: 3031
It took 0.0077114105224609375 seconds.
```