Cryptography

Lab 5

1. Perform a timing-attack on the following RSA implementation.
2. Imlement blind signature scheme [1].
3. Modify decryption algorithm in such a way that it is computed on random (blinded) input.

Naive RSA

```python
import math
import random
from sympy import randprime, isprime, Mod

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('does not exist')
    else:
        return x % m


def GenModulus(w):
    n = len(w) // 2
    p = randprime(2 ** n, 2 ** (n+1))
    q = randprime(2 ** n, 2 ** (n+1))
    N = p * q
    return N, p, q

def GenRSA(w):
    n = len(w)
    N, p, q = GenModulus(w)
    m = (p-1) * (q-1)
    e = 2 ** 16 + 1
    d = modinv(e, m)
    return N, e, d, p, q

def enc(x, N, e):
    return x ** e % N

def dec(c, N, d):
    return c ** d % N

def fast_pow(c, N, d):
    d_bin = "{0:b}".format(d)
    d_len = len(d_bin)
    reductions = 0
    h = 0
    x = c
    for j in range(1, d_len):
        x, r = mod_reduce(x ** 2, N)
        reductions = reductions + r
        if d_bin[j] == "1":
            x, r = mod_reduce(x * c, N)
            reductions = reductions + r
            h = h + 1
    return x, h, reductions

def mod_reduce(a, b):
    reducions = 0
    if a >= b:
        a = a % b
        reductions = 1
    return a, reductions
```

References

[1] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.