Neighbor RSA

main part:

```
e = 65537
p = random_prime(1<<2048)
q = random_prime(1<<512)
r = random_prime(1<<512)
n1 = p * q
n2 = next_prime(p) * r
assert m1 < n1 and m2 < n2
```

Since that we notice that the special generation of `p, q`, we find that this is an `agcd` problem since that `agcd` is like `xi = p*qi+r  for 1<=i<=t, where ri is small. Given some xi, solve for common divisor p.` Using what I have written in my repo on github `/cryptography/agcd` can solve it with `rho=512` since `r` is 512 bits.

Sexy RSA

main part:

```
def getSexyPrime(n=512):
    # Sexy prime: https://en.wikipedia.org/wiki/Sexy_prime
    while True:
        p = getPrime(n)
        if isPrime(p+6):
            return p, p+6
```

Notice this function, we can find that `p,q` are really near. So just use `fermat method` to factorize n.

Proth RSA

In this challenge, we have:

```
def getProthPrime(n=512):
    # Proth prime: https://en.wikipedia.org/wiki/Proth_prime
    while True:
        k = getRandomInteger(n)
        p = (2*k + 1) * (1<<n) + 1
        if isPrime(p):
            return p, k
```

and we also know this:

```
s = (k1 * k2) % n
```

hmmm looks like we have only 2 unknown vars: `k1, k2`. So we need try to solve them over `Zmod(n)`. What came my mind is using `groebner basis` since this is really fast and helpful when we are trying to solve complex equations over polynomial ring. So I just use `k1, k2` to represent `p, q` and then construct:

$$p * q - n$$
$$k1 * k2 - s$$

And then solve for gb. But unfortunately it just gives me something like $k_2^2 + ak_2 + b$. But don't worry, just use `small_roots` to solve for `k2`. After solving for `k2`, just recover `q`.

Leaky RSA

In this challenge, we are given the bit of `p` and `q`, also gives us `s = ((p**3 - 20211219*q) * inverse(p*p+q*q,n)) % n`. Seeing this, I know that this is a challenge to let us using `coppersmith method` to recover `p,q`. In this way, we have to clear `s`.

$$s = (p^3 - aq)(p^2 + q^2)^{-1} \, mod \, n$$
$$=> (p^2 + q^2)s = (p^3 - aq) \, mod \, n$$
$$=> p^3 - aq - p^2 s - q^2 s = 0 \, mod \, n$$
$$=> -aq - q^2 s = 0 \, mod \, p$$
$$=> aq + q^2 s = 0 \, mod \, p$$

So in this way, we construct a polynomial and use `small_roots` to solve for `q`. After getting `q`, we can just encrypt over `Zmod(q^2)` with out solving for `p`. Little trick make us to be faster than solving for `p` lol (but only when the message has no padding)