square-power

March 1, 2025

[2]:
```python
#%%
from Crypto.Util.number import getStrongPrime
from math import gcd
from random import randint
from typing import Tuple
from Crypto.Cipher import AES
from hashlib import sha256

def encrypt(m: bytes, secret_key: int) -> str:
    hash_secret_key = sha256(str(secret_key).encode()).digest()
    cipher = AES.new(hash_secret_key, AES.MODE_ECB)
    return cipher.encrypt(m).hex()

def generate_primes() -> int:
    p = getStrongPrime(512)
    q = getStrongPrime(512)

    while gcd(p*q, (p-1)*(q-1)) != 1:
        print("loop")
        p = getStrongPrime(512)
        q = getStrongPrime(512)

    return p*q

def generate_public_key() -> Tuple[int, int]:
    n = generate_primes()
    k = randint(2, n-1)
    while gcd(k, n) != 1:
        k = randint(2, n-1)
    g = 1 + k * n
    return n, g, k

flag = b"PWNME{xxxxxxxxxxxxxxxxxxxxxxxxx}"

#%%
```

```
n =␣
  ↪1304800012647955112049529819705547652866282820977084975738055624957617469566892948374779247
g =␣
  ↪14232999694821698106937459755169111250723143832548091913379257481041382160905011536064172867
k =␣
  ↪109081848228506024782212502305948797716572300830339785578465230204043919222714279516643240
A =␣
  ↪10331979810348166693003506393334562363373083416444082955583854323636220335613638441209816437
B =␣
  ↪40813422673230181662496076889783806652414238169578757471253288109585906561539737877832468677
enc = "abd9dd2798f4c17b9de4556da160bd42b1a5e3a331b9358ffb11e7c7b3120ed3"
```

```
n, g, k = generate_public_key()

a = randint(2, n-1)
b = randint(2, n-1)


A = pow(g, a, n*n)
B = pow(g, b, n*n)

secret_key = pow(B, a, n*n)

print(f"{n = }")
print(f"{g = }")
print(f"{k = }")

print(f"{A = }")
print(f"{B = }")

print(f'enc = "{encrypt(flag, secret_key)}"')
```

Facts:

This looks like the basis of pallier crypto system. Imporant here is that

$(1+x)^n == 1 + nx \mod n^2$

$g^a == (1+kn)^a == 1 + kna \mod n^2$, we can now continue to modify it until we end up with na

[17]:
```
k_inv = pow(k,-1,n*n)
k_inv
```

[17]:
```
48576754139702720479669405265431283521957174899654076731670992195640317283185952
21225021183114343870037743411391464725683991349857730550428310932192770278357418
01039391536885727553261740761445202881129545278264008006566096799596061665295866
29392048245848995354728020815711676343685992244524368403117081214442217975690295
27960956925669522518869378600233841768936064235818915004377108251276779981513199
78179421920123620814951111948293994414591375809207642512428692208584056583547292
```

```
406387989881497986260168236970686551578205044902512202980216515078057380316019706270235665045720716374938350027249121333553453024418915
```

```
[18]: na = ((A - 1) * k_inv) % (n*n)
      nb = ((B - 1) * k_inv) % (n*n)
      na,nb
```

```
[18]: (15941053848593865955474772420365230460198990088333118155070289089652222903362178508211659120836658700802585227161552745216260286604097617219280315875926579120785403746798455319081841159920791748108998148946627489840340310818771922723878026051538529741054074821181636644358209004284781528470779535557339161054410765666819037617033093399129017478153392569283276487885443195354419372328953154719844602430566954767116098956113024415124369868639677919269146708798358662513431121228236962626801739810079823243518732431674802145556434174164160872767923406835928576545475011553554967655771112144408833647296244189664973387943,
 41322580246334583170172907122323724428094361191299157214058280538048472288264541210237069073730169149726281354624470723880297100143985768618152737546711347571525438181532844381623346785884396857162194593619771076669784755622261256844982812994749730739218330975925919085651471835077438320714809329316832733707047422443451402805675858413891780011936724011557051738551994118711384534405099047865399877641737671985134032299125652497784885212979323848092405177654627954639102619146547502582940340666597366089729072592033546149116164034334449195988797466747362029693404414174880544187404138287489161593362910210770395751)
```

Now, we have

$n * a \mod n^2 == n * a - n^2 * k$ where we can simply divide by n to obtain a because n is on both sides such that we end up with $a \mod n$:)

```
[21]: a = na//n
      b = nb//n

      A = pow(g, a, n*n)
      B = pow(g, b, n*n)

      secret_key = pow(B, a, n*n)
```

```
[22]: hash_secret_key = sha256(str(secret_key).encode()).digest()
      cipher = AES.new(hash_secret_key, AES.MODE_ECB)
      cipher.decrypt(bytes.
       ↪fromhex("abd9dd2798f4c17b9de4556da160bd42b1a5e3a331b9358ffb11e7c7b3120ed3"))
```

```
[22]: b'PWNME{Thi5_1s_H0w_pAl1ier_WorKs}'
```

```
[ ]:
```