Crypto

Block Cipher

task.py 里可以看出来加密主要是使用异或，再异或一次就可以得到密文。

```
import operator
from functools import reduce


def xor(a, b):
    assert len(a) == len(b)
    return bytes(map(operator.xor, a, b))


def decrypt(iv, key, results):
    parts = []
    for index, part in enumerate(results):
        parts.append(reduce(xor, [part, iv if index == 0 else results[index -
1], key]))
    return parts


iv = b'Up\x14\x98r\x14%\xb9'
key = b'\r\xe8\xb86\x9c33^'
results = [b'0\xff\xcd\xc3\x8b\\T\x8b', b'RT\x1e\x89t&\x17\xbd',
b'\x1a\xee\x8d\xd6\x9b>w\x8c', b'9CT\xb3^pF\xd0']
flag = []
flag = decrypt(iv, key, results)
```

Multi Prime RSA

多素数因子 RSA ，根据中国剩余定理~~【根本不会】~~，在网上找了个解法大概理解了一下。

```
import gmpy2

p =
617899321487194773840274583333805689780562861361378290929523173077119083534 77
q =
912079693533557636856332843788335063197947145070273329292907017487275341938 61
r =
```

```
105471299607375388622347272479207944509670502835651250945203397530010861809367
s =
831532387489037724481383075055797992771626521512444773914651305042671718814 37
n =
103934437216508710000106392059815181232415106468484184525097475852526514856770610378495842487318172135244020928481249375397255651948202632728264461909146688652380484124827721035317338340794459845384811381586690859533561945854948695876449010380847532959808584218496306506849948988646791108729508716376259928462205518545690577450724578166729319920531769202982949596148734794481387441542377198066077898621114584171241263115636912914647011913513637815820345957659624616919141948856083273404607610767309199586002186323988260863845814993025594418486380127838655103198014646023151574775441167865175269888100146497398142424078141308494194726187528972553895972057249632934849987058005799754084448830911105924074508104832476286657294837122283927871803443573982767719002550080245362687235620861271841724964947457119716707691640358239418635781264056625093036127622996955312812831273624544012955602010818883596613142595643179641772043647409338177079643162952305437825849754601322249497454926214041558515898594096641545947815072283211969130869751018902644735918999405588509073541173833229625401120854767691400486473232786388421773345628736977108709451470846868564182037522083548505348257085261936309117332420333450346182398361088684993094425055392885550601268450421152554299857527562678412973634514277239910927361952244591 9
e = 65537
c =
844677395496466411520394190869787261209960246734415406217975986418865760680024542119231873259131861208878522030009923057991526761346423130242121884493257732067700857897379859545356609151834223804262174935191718271211809221730601602827122249238086030580971376104724987801049500689134122609834321586609223761140538079460830213824674361601046367637227094018381901291488659642720549583856812747877519600804325570421770575999289389175021646347371879234023647657507178519047236746071420327155188213839293382288787853777540226192644761028822256165706787395891134765908229036044468473519166141610604791485071702808854944672418124203289328124793348198048601338476086482318248264508789781967910205393740835345086784345145351367491197717933757414967811594913692588314161669333147733048171044386546892346475181197482702164468542430187885074163177843285948999943328049159021873821254267471067523609151007885131921896462161216356454116929796355815756642621369974260365378070336290542971599886325232821981080341858950609157813769416455337935096696635623426418166316737131174435618543058086342714723330814586496030805366321181723292731710369013923285787724941830672247377301048663929453294620044701627159066468762709113137517559435822623284148112827473010030736329596829357275518641576798298066541516764673029908084962144713

dp = gmpy2.invert(e, p - 1)
dq = gmpy2.invert(e, q - 1)
dr = gmpy2.invert(e, r - 1)
ds = gmpy2.invert(e, s - 1)

mp = pow(c, dp, p)
mq = pow(c, dq, q)
```

```
mr = pow(c, dr, r)
ms = pow(c, ds, s)

qInv1 = gmpy2.invert(q, p)
h1 = (qInv1 * ((mp - mq) % p)) % p
m1 = mq + h1 * q

qInv2 = gmpy2.invert(s, r)
h2 = (qInv2 * ((mr - ms) % r)) % r
m2 = ms + h2 * s

p1 = p * q
p2 = r * s
qInv2 = gmpy2.invert(p2, p1)
h3 = (qInv2 * ((m1 - m2) % p1)) % p1
m = m2 + h3 * p2
print(hex(m))
```

RSA Attack 3

e 很大，低解密指数攻击，直接从网上学习~~【复制】~~了一个脚本。

```
#!/usr/bin/python
# coding:utf-8

import gmpy2
from Crypto.PublicKey import RSA
import ContinuedFractions, Arithmetic
from Crypto.Util.number import long_to_bytes


def wiener_hack(e, n):
    # firstly git clone https://github.com/pablocelayes/rsa-wiener-attack.git
!
 frac = ContinuedFractions.rational_to_contfrac(e, n)
    convergents = ContinuedFractions.convergents_from_contfrac(frac)
    for (k, d) in convergents:
        if k != 0 and (e * d - 1) % k == 0:
            phi = (e * d - 1) // k
            s = n - phi + 1
 discr = s * s - 4 * n
            if (discr >= 0):
                t = Arithmetic.is_perfect_square(discr)
                if t != -1 and (s + t) % 2 == 0:
                    print("Hacked!")
                    return d
```

```
    return False


def main():
    n =
507419170088344932990702256911694788408493968749527614421614568612944144764889717229444020813658893362983714454159980719026366361318789415279417172858536381938870379267670180128174798344744371725609827872339512302232610590888649555446972990419313445687852636305518801236132032618350847705234643521557851434711389664130274468354405273873218264222293858509477860634889001898462547712800153111774564939279190835857445378261920532206352364005840238252284065587291779196975457288580812526597185332036342330147250312262816994625317482869849388424397437470502449815132000588425028055964432298176942124697105509057090546600330760364385753313923003549670107599757996810939165300581847068233156887269181096893089415302163770884312255957584660964506028002922164767453287973102961910781312351686488047510932997937700597992705557881172640175117476017503918294534205898046483981707558521558992058512940087192655700351675718815723840568640509355338482631416345193176708501897458649841539192993142790402734898948352382350766125000186026261167277014748183012844440603384989647664190074853086693408529737767147592432979469020671772152652865219092597717869942730499507426269170189547020660681363276871874469322437194397171763927907099922324375991793759
    e =
773101998674486777820815721093434727837811356417125976435971225914430112290915335167589252389497554913954894089224374936702525509208266414421896839079739268435054367300148999185874779130322861535452470634938859829411949962517998829841451557330500695644851206607161108281107387846442235197256132801400067836183939951380760306164633982848195506276121020102143152352699452517414078996922749786426636506871577364178312904048711819024639043110954483684984321472929388254189305271887206964975968675758434768102251526592445294814809938431683830165830687477331187030002874233740940518957244941934551751311202430970652708044577870264925789165845368635484458139168194178570640376641016844550001849875312523445828995897462721739700837331301064078106192580772666038985292856344957108468380118582870243295144910587905573050413896146507302677744829546667269498863133868810665939467894600283995232457771713203194446735512683791262038625766275401778882902657144180643347524999405877503745523300081437085620659402456376858337134860333883444721224864886951458504787144206041262216427689476623838389469375934759097792630658108039068536061540776660057352756501691483013206642845473813538017895959069214557741881167763905092979199631318029792483369009535
    c =
165251729917394529793163344300848992394021337429474789711805041655116845722480301677817165053253655027459227404782607373107477419083333844871948673626672704233977397989843349633720167495862807995411682262559392496273163155214888276398332204954185252030616473235814999366132031184631541209554169938146205402400412307638567132128690379079483633171535375278689326189057930259534983374296873110199636558962144635514392282351103900375366360933088605794654279480277782805401749872568584335215630740265944133347038070337891035560658434763924576508969938866566235926587685108811154229747423410476421860059769485356567301897413767088823807510568561254627099309752215808220067495561412081320541540679503
```

```
218232020279947159175547517811501280846596226165148013762293861131544331444165070186672186027410082671602892508739473724143698396105392623164025712124329254933353509384748403154342322725203183050328143736631333990445537119855865348221215277608372952942702104088940952142851523651639574409075484106857403651453121036577767672430612728022444370874223001778580387635197325043524719396707713385963432915855227152371800527536048555551237729690663544828830627192867570345853910196397851763591543484023134551876591248557980182981967782409054277224
```

```
 d = wiener_hack(e, n)
    m = pow(c, d, n)
    print(long_to_bytes(m))


if __name__ == "__main__":
    main()
```