密码学硬核笔记——扩展维纳攻击

原创

Gm1y 已于 2022-02-11 21:53:39 修改 1921 收藏 15

分类专栏： 原题复现 密码学硬核笔记 文章标签： 密码学 算法

于 2020-11-17 16:10:31 首次发布

Preface

打羊城杯遇到了Extending Wiener's attack，发现自己的知识面还是太浅了。因此写一篇博客来学习一下。

Wiener's attack

Wiener's attack 是根据连分数的性质以及定理，通过对e/N的连续逼近找到对应的k/d，从而分解N。接下来讲一下连分数的定义

Continued Fractions（连分数）

连分数就是一个数的连续分式展开，它的式子长这样：

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}},$$

（其中a_0 是整数，a_1,a_2,…,a_n都是正整数）

通常我们用更简单的数组方式来描述,对任何有理数p/q来说：

$$p = [a_0; a_1, a_2, \ldots, a_n]$$

计算连分数我们可以使用欧几里得算法：

$$
\begin{aligned}
p &= a_0 q + r_0 \\
q &= a_1 r_0 + r_1 \\
r_1 &= a_2 r_1 + r_2 \\
&\vdots \\
r_{n-2} &= a_n r_{n-1} + 0.
\end{aligned}
$$

Convergent(收敛)

我们定义 c 为连分数每一次分式展开的收敛，即：

$$\forall i \in [0,n], c = [a_0, \ldots, a_i]$$

Legendre's theorem

这是Wiener's attack 的一个比较重要的定理，定义如下：

Theorem 2.2 (Continued Fractions). *Let $\alpha \in \mathbb{Q}$ and $c, d \in \mathbb{Z}$ satisfy*

$$\left|\alpha - \frac{c}{d}\right| < \frac{1}{2d^2}.$$

Then c/d, in lowest terms, is one of the convergents in the continued fraction expansion of α.

也就是说，当满足 $|\alpha - \frac{c}{d}| < \frac{1}{2d^2}$ 时，$\frac{c}{d}$ 就是 α 的连分数收敛。

根据这个定理，我们可以通过计算一个数（e/N）的连分数来找到与这个数近似的两个数的比值(k/d)，这就是Wiener's attack的攻击方法。

Applied to RSA

假设:

$ed = 1 + k\lambda(N)$

$\lambda(N) = LCM(p-1, q-1)$

$q = gcd(p-1, \ldots)$

当 $d < N$ 时，可算得：

那怎么通过 得到 p 和 q 呢？

我们已知：

$$edg = k(p-1)(q-1) + \ldots$$

Sagemath实现：

```
#wiener_attack
def possible(e,alist,N):
    for x in alist:
        if x==0:
            continue
        phi = floor(e*(1/x))
        if (N-phi+1)%2==0 and sqrt(pow((N-phi+1)//2,2)-N).is_integer():
                (p,q)=var('p,q')
                x=solve([(p-1)*(q-1)==phi, p*q==N],p,q)
                return int(str(x[0][0]).split('==')[1])
        else:
            continue

def wiener_attack(e,N):
    c=continued_fraction(e/N)
    alist=c.convergents()
    return possible(e,alist,N)

n=
e=
t=wiener_attack(e,n)
print t
```

RRRRRRSA

其实这里我还想提一下 $Legendre\ s\ theorem$，因为这个定理我们可以当成一个工具，而不是仅仅用在Wiener ' s attack 。 这个这个定理的关键在于可以将一个模糊近似的数具体地求出来，类似gmpy2.iroot()一样。可以作为一种尝试的工具。（只要你有办法构造出来）

这次2020羊城杯的RRRRRRSA

```
P1 = getPrime(1038)
P2 = sympy.nextprime(P1)
assert(P2 - P1 < 1000)

Q1 = getPrime(512)
Q2 = sympy.nextprime(Q1

N1 = P1 * P1 * Q1
N2 = P2 * P2 * Q2
```

已知N1,N2，其他都是未知的。tips里也说了需要用到连分数。那必然是要用到这个定理。

下图是对应的论文。

Proof. Suppose that $N_1 = p_1^r q_1$ and $N_2 = p_2^r q_2$ with $p_1 > p_2$. Then $q_2 N_1 - q_1 N_2 = q_1 q_2 (p_1^r - p_2^r)$. Hence

$$\left| \frac{N_2}{N_1} - \frac{q_2}{q_1} \right| = \frac{q_1 q_2 \left| p_1^r - p_2^r \right|}{q_1^2 p_1^r}.$$

In order to apply Theorem 2, we need that $\frac{q_1 q_2 |p_1^r - p_2^r|}{q_1^2 p_1^r} < \frac{1}{2q_1^2}$, or equivalently

$$|p_1^r - p_2^r| < \frac{p_1^r}{2q_1 q_2}. \tag{1}$$

通过 的连分数来找q。

Extending Wiener 's attack

Wiener 's Approach

$$edg = k(p - \quad -1) +$$

所以

$$edg - kN = q + k_0(W)$$

Guo 's Approach With Two Exponents

假设对于一个N来说有两个e，而且都有相对应比较小的d，那么可以得到：

$$\begin{cases}$$

所以可推出

$$|$$

那么就可以通过求 的连分数找到

但是通过

Extending Wiener 's attack

Howgrave-Graham 和 Jean-Pierre Seifert 结合了Wiener 和Guo的想法，通过多个等式来构造格利用LLL化简来解决这个问题。

首先假设$d < N$,联立四个等式：

（1）

$$k\ k = k\ k$$

（2） $*k$

（3） G

$$e\ d\ k - e\ d\ k = k - k$$

（4） $W\ W$

从而构造出一个都是已知数的矩阵 L

$$A = ($$

$$L = \quad -e\ e\ 0$$

$$B = ($$

$$*\ L = B$$

根据假设可知：

根据 $Minkowoski\ s\ first\ theorem$，只有满足：

$$\lambda\ (L) \leq$$

但是右边计算结果为 N，明显 $||B|| > N$，所以不行。

其实，我们可以通过对矩阵的列乘上一个倍数使得条件满足。

令

$M = N$ ，

$$L = \quad ,$$

所以，要使得不等式成立，可推得：

$$\|B\| \leq \quad \det(L)$$

$$2N \quad \leq 2N$$

求出了α，我们可以代入M，利用LLL算法求出格L的最短向量B,再乘上L的逆求出A，利用A的前两项求出$\varphi(N)$。

真题演练

2020羊城杯 Simple.py

```
from Crypto.Util.number import *
from Crypto.Cipher import DES
import gmpy2
from secret import flag
import random

key = "abcdefgh"

def des_encrypt(m):
    des = DES.new(key, DES.MODE_ECB)
    res = des.encrypt(m)
    return res

def gen_key():
    p = getPrime(2048)
    q = getPrime(2048)
    n = p * q
    bit = n.bit_length()
    phi_n = (p - 1) * (q - 1)
    num = random.randint(1, 100)
    while True:
        u = getPrime(bit / 4 - num)
        if gmpy2.gcd(u, phi_n) != 1:
            continue
        t = gmpy2.invert(u, phi_n)
        e = bytes_to_long(des_encrypt(long_to_bytes(t)))
        if gmpy2.gcd(e, phi_n) == 1:
            break
    return (n, e)

P = getPrime(1024)
Q = getPrime(1024)
N = P * Q
E = 65537
lcm = gmpy2.lcm(P-1, Q-1)
e1 = gmpy2.invert(getPrime(730), lcm)
e2 = gmpy2.invert(getPrime(730), lcm)
m = bytes_to_long(flag)
```

```
c = pow(m, E, N)
print "N = " + str(N)
print "e2 = " + str(e2)
print "c = " + str(c)
_n, _e = gen_key()
_c = pow(e1, _e, _n)
print "_n = " + str(_n)
print "_e = " + str(_e)
print "_c = " + str(_c)

# N = 14922959775784066499316528935316325825140011208871830627653191549546959775167708525042423039865322548420928
5715241207438316935501235634939817979509128958934762004470833865493533360868990649218785820743467913201041061399
6501048061487959235779305334257785076110894408631847584988244027268824681802220935685292421523748146022937754429
7224983887026669222885987323082324044645883070916243439521809702674295469253723616677245762242494478587807402688
4741761020934820194171187037474118624205362406110895293311486844405139346094128849410916515948615306060869821748
62461739604705354416587503836130151492937714365614194583664241
# e2 = 2718882573172758465662471298870315103012635053615747759193555850881772258034368956592432944215123964960799
3377452763119541243174650065563589438911911135278704499670302489754540301886312489410648471922645773506837251600
2441096198501417627959016965033878800586580614905950342818840892654873363730114248834044991240024418608702912338
7504567521235528762294842710936292519901838353525991354985974715834893184704190791031346553170381031347267443542
5886505383646969400166213185676876969805238803587967334447878968225219769481841748776108219650785975942208190380
614555719233460250841332020054797811415069533137170950762289
# c = 64723673388326359068964239903235425376638493043141715815541074952108300266602116960890629161588941955617230
4786460463346043386783868733837067628716027416591580023525364069051004606654144514050191773102659642708055856736
6267665887665459901724487706983166070740324307268574128474775026837827907818762764766069631267853742422247229582
7562562531759418990998988846563345987907113793054904199326641146150103820945728547994218916227896146147204427082
7165337648566013956081966823911858806931217929348868440340438571578040693781712458877368992164280270300534132400
8483201528345805611493251791950304129082313093168732415486813
# _n = 4404892382649008607769490638452005587343411822539110401046897266344144889970955182849645140780799118563528
2417417393725155884225134976263171679830736099541454546451435595749946039635245634105832967147038449354704218223
8690727766731554287411757022792467324815342497916894285866240516524768645049867582541899123632009100512965460004
5483820545784612499901584426752344771225211896493163416236371468675891199518313857175139649417875620688915230608
4317046360025551872807095850922405346004118486994303888743443502442831106353334551482782748512105502224580082372
3487812635502090530820946638405345755666124356919178290008475459419571761406117827422883820901663916276191422633
9406991137605161490026096722306105754426438222411268242877900552641627252091201926619852594239243077854520019277
0132364724778265877578011764290069483147568103763469180623221128649318712146450612201288964413736407940318335377
4265910554863733455161820449073656744610495110838881353269890437984975607744603113572453211439334880155671730821
7553610547812436394079121339715303940319337850517707253312429329292447195948305483107689370370422437945511638914
5154557483783835739807263870990795821606799989184239537695359694037745730832933652448896253262085023757027913456
7668379
# _e = 8616056548522366684140103860167827297455494777229019709332203804526520520185021137379682045297904957392332
5857220942277425713925636792864955456256188901316434460826955577715044665117069725538134443728300350847633681413
2594917061838422072660017477530465048729471603537912401826065081663165440462979219418291010867656746870617893935
7582415910323500107828619887428859180155324940204063508970485751558009419911079734339155730302550704110737934892
1878286222592146529505590768973441388126317902974187052079781628242023009087968728757532829417144881980353020529
2587159921154471289747571107461754730577787617451127061265552788125691266357724955508391085485034126227212788895
4169021894795871949998187646394037525961650438832955064659162777344823802523995573956215664613226645593444838891
8703785117843101122013491456043865752278740963267702026908689514248866920346925662917343831348704613023801020667
8820035631793666627274457756812810094004185303422637897314225624079032617334487815628021058997628511963565055629
4352789562518693290255446232912239841905621091493161592435653235652714913563781895610050846765927864535814313936
5138518132652545544115596043294668297651575616103829331343386207876300470400335698337178741478710407640112144438
3911561
# _c = 3059378395465944392304638615846042010773747591674684108278309435284030079417796588816724777051136176148286
1133242719912421788793739137828194385615957105759820370936689154740197432601698071113019727531214996610515157374
8299654404630150641461765232935912266448303266990247145252052886920248198006212876273661195636104435277145396636
9855160641545344887508794534742118524614630419608357456953685779037867026075084926585632721210386933717522890173
3078171923575201869763530445832100840793098656577982627804808276475436726746063779851278015328132573334899942640
7049795270044819657399403071013496169060640127279409914638535996355848933378734045908205536540619564723586905257
5694987167078205443510923795164659435373834226803573338492481291181485433897333956863995659995868991230873100254
```

```
42994131218237679518267106194962305629529210402269726736072967966518381350920965727690274018080619332676536005722214955949897632990356174168234408837737546230730400434240785496100281815168806724358191550743656843853383646410487436540166360406982096949178466861150173527305369007546917550634679211293496458282787881244581230558011582720632502886494712233308474151958909251857281750741736910202763888790654287328846201724930302778996046434656839999091303411
```

首先这题分为两部分。第一部分是gen_key()以及后面的输出求出e1。第二部分就是利用e1,e2求出c。

第一部分的t可以直接DES解密求出来，然后又因为u满足

$u< N$ 的条件，所以可以直接用wiener atttack分解出_n，从而求出e1。

```
from Crypto.Util.number import *
from Crypto.Cipher import DES
import random
from gmpy2 import invert

key = "abcdefgh"

def des_decrypt(m):
    des = DES.new(key, DES.MODE_ECB)
    res = des.decrypt(m)
    return res

#gkd之无敌究极wiener_attack
def possible_phi(e,alist,N):
    for x in alist:
        if x==0:
            continue
        phi = floor(e*(1/x))
        if (N-phi+1)%2==0 and sqrt(pow((N-phi+1)//2,2)-N).is_integer():
            (p,q)=var('p,q')
            x=solve([(p-1)*(q-1)==phi, p*q==N],p,q)
            return int(str(x[0][0]).split('==')[1])
        else:
            continue

def wiener_attack(e,N):
    c=continued_fraction(e/N)
    alist=c.convergents()
    return possible_phi(e,alist,N)
_n = 4404892382649008607769490638452005587343411822539110401046897266344144889970955182849645140780799118563528241741739372515588422513497626317167983073609954145454645143559574994603963524563410583296714703844935470421822386907277667315542874117570227924673248153424979168942858662405165247686450498675825418991236320091005129654600045483820545784612499901584426752344771225211896493163416236371468675891199518313857175139649417875620688915230608431704636002555187280709585092240534600411848699430388874344350244283110635333455148278274851210550222458008237234878126355020905308209466384053457556661243569191782900084754594195717614061178274228838209016639162761914226339406991137605161490026096722306105754426438222411268242877900552641627252091201926619852594239243077854520019277013236472477826587757801176429006948314756810376346918062322112864931871214645061220128896441373640794031833537742659105548637334551618204490736567446104951108388813532698904379849756077446031135724532114393348801556717308217553610547812436394079121339715303940319337850517707253312429329292447195948305483107689370370422437945511638914515455748378383573980726387099079582160679998918423953769535969403774573083293365244889625326208502375702791345676768379
_e = 86160565485223666841401038601678272974554947772290197093322038045265205201850211373796820452979049573923325857220942277425713925636792864955456256188901316434460826955577715044665117069725538134443728300350847633681413259491706183842207266001747753046504872947160353791240182606508166316544046297921941829101086765674687061789393575824159103235001078286198874288591801553249402040635089704857515580094199110797343391557303025507041107379348921878286222592146529505590768973441388126317902974187052079781628242023009087968728757532829417144881980353020529258715992115447128974757110746175473057778761745112706126555278812569126635772495550839108548503412622721278889541690218947958719499981876463940375259616504388329550646591627773448238025239955739562156646132266455934448388918
```

```
037851178431011220134914560438657522787409632677020269086895142488669203469256629173438313487046130238010206678820035631793666627274457756812810094004185303422637897314225624079032617334487815628021058997628511963565056294352789562518693290255446232912239841905621091493161592435653235652714913563781895610050846765927864535814313936513851813265254554411559604329466829765157561610382933134338620787630047040033569833717874147871040764011214443839
11561
_c = 30593783954659443923046386158460420107737475916746841082783094352840300794177965888167247770511361761482861133242719912421788793739137828194385615957105759820370936689154740197432601698071113019727531214996610515157374829965440463015064146176523293591226644830326699024714525205288692024819800621287627366119563610443527714539663698551606415453448875087945347421185246146304196083574569536857790378670260750849265856327212103869337175228901733078171923575201869763530445832100840793098656577982627804808276475436726746063779851278015328132573334899942640704979527004481965739940307101349616906064012727940991463853599635584893337873404590820553654061956472358690525756949871670782054435109237951646594353738342268035733384924812911814854338973339568639956599958689912308731002544299413121823767951826710619496230562952921040226972673607296796651838135092096572769027401808061933267653600572221495594989763299035617416823440883773754623073040043424078549610028181516880672435819155074365684385338364641048743654016636040698209694917846686115017352730536900754691755063467921129349645828278788124458123055801158272063250288649471223330847415195890925185728175074173691020276388879065428732884620172493030277899604643465683999909130341
_t = bytes_to_long(des_decrypt(long_to_bytes(_e)))
_p = wiener_attack(_t,_n)
_q = _n//_p
_d = invert(_e,(_p-1)*(_q-1))
e1 = pow(_c,_d,_n)
print e1
```

第二部分就直接用Extending Wiener Attack求解即可。

```
from Crypto.Util.number import *
from gmpy2 import invert
c = 64723673388326359068964239903235425376638493043141715815541074952108300266602116960890629161588941955617230478646046334604338678386873383706762871602741659158002352536406905100460665414451405019177310265964270805585673662676658876654599017244877069831660707403243072685741284747750268378279078187627647660696312678537424222472295827562562531759418990998988846563345987907113793054904199326641146150103820945728547994218916227896146147204427082716533764856601395608196682391185880693121792934886844034043857157804069378171245887736899216428027030053413240084832015283458056114932517919503041290823130931687324154868132015283458056114932517919503041290823130931687324154868132
e2 = 27188825731727584656624712988703151030126350536157477591935558508817722580343689565924329442151239649607993377452763119541243174650065563589438911911135278704499670302489754540301886312489410648471922645773506837251600244109619850141762795901696503387880058658061490595034281884089265487336373011424883404499124002441860870291233875045675212355287622948427109362925199018383535259913549859747158348931847041907910313465531703810313472674435425886505383646969400166213185676876969805238803587967334447878968225219769481841748776108219650785975942208190380614555719233460250841332020054797811415069533137170950762289
e1 = 11455245955373035796101326833369887965900791903594293031343280977679966918148166030653124361816012792230426498600150178456457512831988499177454268285346680832997336201967728407264667828005109196455561122096171930232054740588038611351914707629948159499779988438401254850624074804236564321277421573030404787167970603559655089894458031492326098276885813339518777702991415006437199832878806888844080356596456766256365206284538837989779950643938946161942293331862576560342360461513721737561209122157833949326316067035503289818679247903477111867839446485441382434730550513562513542881639405307836560393733727179877413895959
N = 14922959775784066499316528935316325825140011208871830627653191549546959775167708525042423039865322548420928571524120743831693550123563493981797950912895893476200447083386549353336086899064921878582074346791320104106139965010480614879592357793053342577850761108944086318475849882440272688246818022209356852924215237481460229377544297224983887026669222885987323082324044645883070916243439521809702674295469253723616677245762242494478587807402688474176102093482019417118703747411862420536240611089529331148684440513934609412884941091651594861530606086982174862461739604705354416587503836130151492937714365614194583664241
a = 0.356#731./2049
M1=N**0.5
M2= N **(a+1)
D = diagonal_matrix(ZZ,[N,M1,M2,1])
M=matrix(ZZ,[[1,-N,0,N**2],[0,e1,-e1,-e1*N],[0,0,e2,-e2*N],[0,0,0,e1*e2]])*D
L=M.LLL()
t=vector(ZZ,L[0])
x=t*M**(-1)
phi = int(x[1]/x[0]*e1)
d = invert(0x10001,phi)
m=pow(c,d,N)
print long_to_bytes(m)
```

References

Extending Wiener’s Attack in the Presence of Many Decrypting Exponents