我是如何破解你的私钥的？或者说，为什么 small keys 不安全？

在以下的博文中，我将解释为什么使用 small RSA keys 公钥是个坏主意。为了有更好的直观性，我将展示因式分解和破解私钥的所需步骤。

什么是 RSA？

RSA 是一种非对称公钥密码体制，这一名称源自发明者 Rivest、Shamir 以及 Adleman 的姓名。为了成功地实现加密和解密，需要两种秘钥。一个密钥对由以下两种秘钥组成：

- 私钥：接收方需通过这把秘钥解密信息，这把钥匙不能公诸于众。
- 公钥：发送方需通过这把秘钥在传输前加密信息，这把钥匙可以公诸于众。

让我们来看一下 RSA key generation 是怎么工作的：

1. 找到两个不同的素数（prime numbers）p 和 q：例如 p = 61 和 q = 53
2. 计算模数（modulus）n=p*q：n=61*53=3233
3. 计算 phi(n)=(p-1)*(q-1)：phi(3233)=(61-1)*(53-1)=60*52=3120
4. 找到一个与 phi(n)互质且满足 1 < e < phi(n)的数 e，有个选择的小诀窍是：选取不能整除 phi(n)的质数 e。我们令 e=17
5. 计算 e 的模乘逆 d: d=2753

*注：d = (k * phi(n) + 1)/e，d 和 k 均是整数。译者使用下面的代码计算 d：*

```
public class Tester
{
    public static void main(String[] args)
    {
        int phi=3120;
        int e=17;
        int k=1;
        int d=0;
        while(true)
        {
            if ((k*phi+1)%e==0)       //d 应为整数，所以做此判断。
            {
                d = (k * phi + 1)/e;
                System.out.println(k);
                System.out.println(d);
                break;
            }
            k++;
        }
    }
}
```

现在，我们获得了钥匙对的所有数字：

- 公钥是（n=3233, e=17）
- 私钥是（n=3233, d=2753）

加/解密 m（message）也是简单的：

- 加密：c(m) = m^e mod n
- 解密：m(c) = c^d mod n

使用 OpenSSL

现在，既然我们知道了 RSA 算法背后的理论，我们可以借用 OpenSSL 做事情了。

让我们来产生一个私钥：

```
# openssl genrsa 128 > my.key
Generating RSA private key, 128 bit long modulus
....++++++++++++++++++++++++++
.++++++++++++++++++++++++++
e is 65537 (0x10001)
```

注：这里的秘钥长度只有 128 位，这么短的秘钥一般只能用于演示或者教学。

genrsa 命令完成了上述的 1-5（也许还多做了些其他的事）。注意到模数（n）只有 128 位长。这通常意味着质数 p 和 q 仅有它一半的长度，因为 n=p*q。

你能通过 rsa 命令来获得一些关于私钥的信息。

```
# openssl rsa -inform PEM -text -noout < my.key
Private-Key: (128 bit)
modulus:
    00:a3:47:4c:ba:d9:37:ee:86:ec:ba:1f:5f:cd:5c:
    14:33
publicExponent: 65537 (0x10001)
privateExponent:
    76:9c:e0:2f:c8:86:ea:3e:6e:fe:74:a6:12:49:07:
    c1
prime1: 14919959527609940627 (0xcf0e58241a653a93)
prime2: 14546579181737124833 (0xc9dfd4b1d888a3e1)
exponent1: 11612102714875570159 (0xa12679e54781e7ef)
exponent2: 4328659950901588545 (0x3c127d4a86f2f241)
coefficient: 4743112561756247098 (0x41d2ebbead2c943a)
```

```
root@kali2:~/桌面/RSA_OpenSSL# openssl rsa -inform PEM -text -noout < my.key
Private-Key: (128 bit)
modulus:
    00:a3:47:4c:ba:d9:37:ee:86:ec:ba:1f:5f:cd:5c:
    14:33
publicExponent: 65537 (0x10001)
privateExponent:
    76:9c:e0:2f:c8:86:ea:3e:6e:fe:74:a6:12:49:07:
    c1
prime1: 14919959527609940627 (0xcf0e58241a653a93)
prime2: 14546579181737124833 (0xc9dfd4b1d888a3e1)
exponent1: 11612102714875570159 (0xa12679e54781e7ef)
exponent2: 4328659950901588545 (0x3c127d4a86f2f241)
coefficient: 4743112561756247098 (0x41d2ebbead2c943a)
root@kali2:~/桌面/RSA_OpenSSL#
```

然而，私钥是我们的秘密，并且我们需要公钥来加密明文。所以我们需要通过-pubout 将公钥从 my.key 提取出来。

```
# openssl rsa -pubout -in my.key > my.pub
writing RSA key
```

```
root@kali2:~/桌面/RSA_OpenSSL# openssl rsa -pubout -in my.key > my.pub
writing RSA key
```

```
# openssl rsa -inform PEM -text -noout -pubin < my.pub
Public-Key: (128 bit)
Modulus:
    00:a3:47:4c:ba:d9:37:ee:86:ec:ba:1f:5f:cd:5c:
    14:33
Exponent: 65537 (0x10001)
```

```
root@kali2:~/桌面/RSA_OpenSSL# openssl rsa -inform PEM -text -noout -pubin < my.pub
Public-Key: (128 bit)
Modulus:
    00:a3:47:4c:ba:d9:37:ee:86:ec:ba:1f:5f:cd:5c:
    14:33
Exponent: 65537 (0x10001)
root@kali2:~/桌面/RSA_OpenSSL#
```

```
# cat my.pub
```

```
root@kali2:~/桌面/RSA_OpenSSL# cat my.pub
-----BEGIN PUBLIC KEY-----
MCwwDQYJKoZIhvcNAQEBBQADGwAwGAIRAKNHTLrZN+6G7LofX81cFDMCAwEAAQ==
-----END PUBLIC KEY-----
root@kali2:~/桌面/RSA_OpenSSL#
```

正如你所看到的，我们的公钥只包含了模数 n 以及指数 e。文件 my.pub 是 PEM 格式的公钥文件。

让我们使用公钥文件对明文进行加密。OpenSSL 的 rsautl 所做的工作如下：

```
# echo -n "Hi"| openssl rsautl -encrypt -inkey my.key > message
```

```
root@kali2:~/桌面/RSA_OpenSSL# echo -n "Hi"| openssl rsautl -encrypt -inkey my.key > message
```

```
# cat message
```

```
root@kali2:~/桌面/RSA_OpenSSL# cat message
▯▯-▯ěE▯wU-▯▯▯root@kali2:~/桌面/RSA_OpenSSL#
```

```
# cat message | hexdump
```

```
root@kali2:~/桌面/RSA_OpenSSL# cat message | hexdump
0000000 af82 807e 97d3 c145 d577 2d8d bcac c7ac
0000010
```

注：以上做法只适用于小于模长度的明文。现在这段用 RSA 加密的明文在以往常常用“对称秘钥”的形式加密。

如你所看的，我们通过公钥对“Hi”进行了加密，其结果是一派“胡言乱语”。

只有接收方使用私钥时才能解密它。

破解私钥

如上述，发送方需要接收方的公钥来进行加密。不幸的是，如果 n 太小了（通常是长度 <1048 位）人们可以（比较轻易地）通过将模数 n 质数分解成两个质数 p*q。因而一个敌手可以破解秘钥并解密消息。

我已经选择了很小的秘钥（它大于 192 位）。让我们假想自己是敌手，并且对解密获取明文感兴趣。我们仅仅拥有公钥，因为这把公钥已经被上传到了公钥服务器，而公钥的密文形式如下：

```
Modulus:
    00:a3:47:4c:ba:d9:37:ee:86:ec:ba:1f:5f:cd:5c:
    14:33
Exponent: 65537 (0x10001)
```

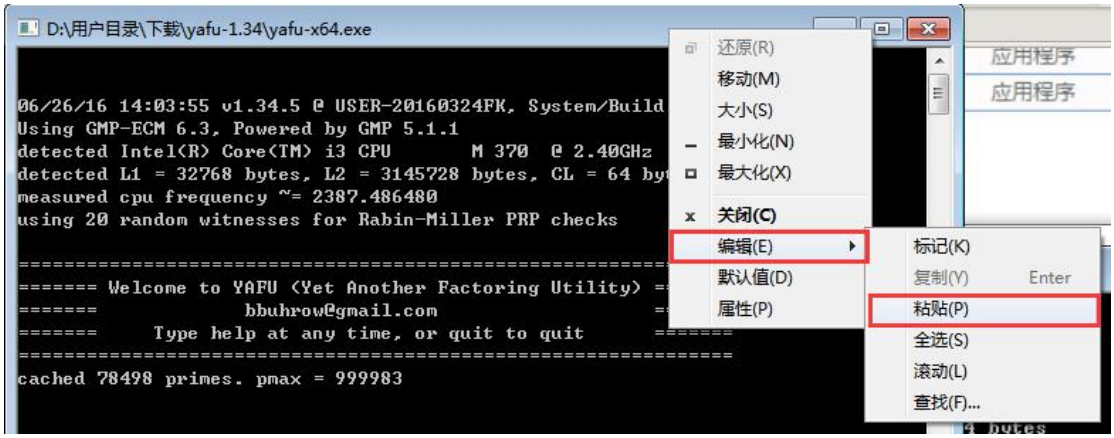
模数 n 是 0x00a3474cbad937ee86ecba1f5fcd5c1433（仅仅简单地把冒号删掉），最耗时的环节是找到两个质数 p’和 q’使得 n=p’*q’。

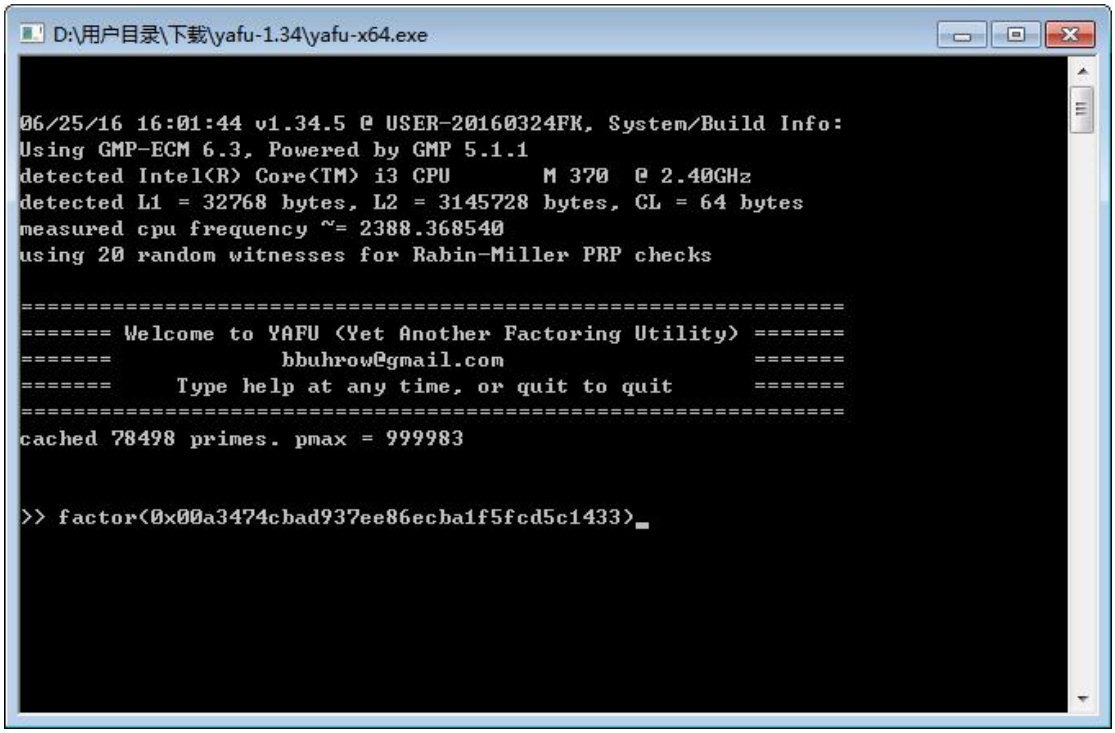
我们可以借用 Yafu 这类工具或者是 factordb.com 这类网站：

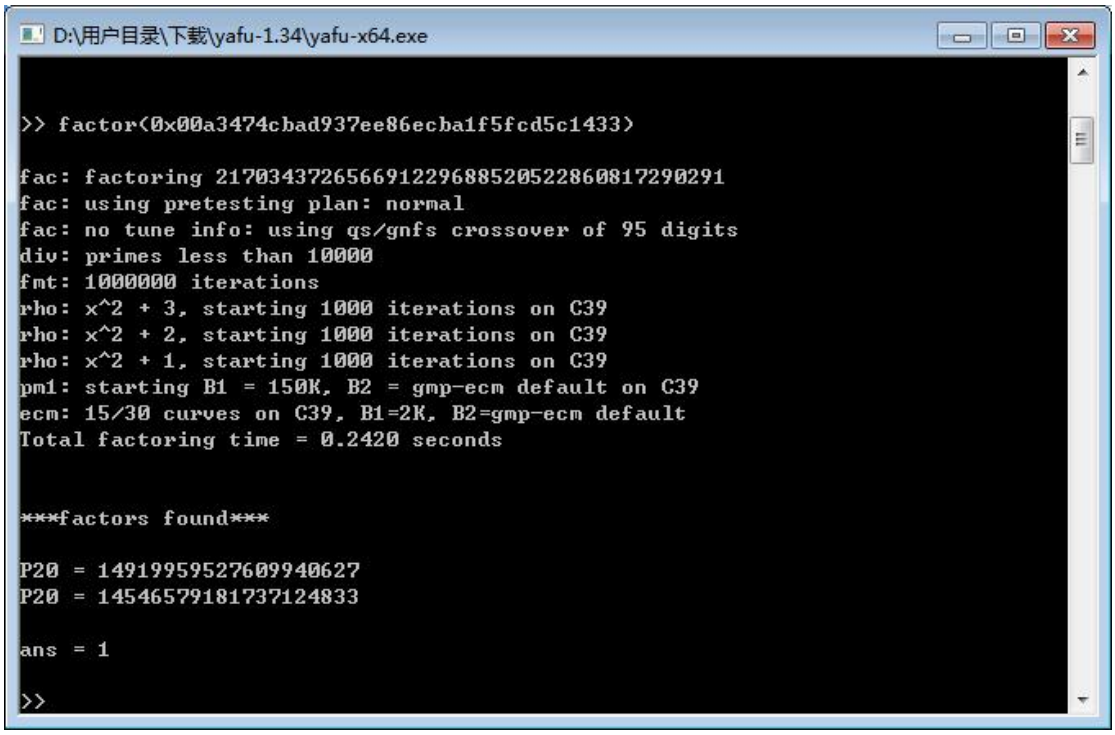
注：我在该网站下载了 Yafu：https://sourceforge.net/projects/yafu/

在“windows 7 64 位”中，解压“yafu-1.34.zip”后，进入文件夹双击“yafu-x64.exe”即可使用。

在命令行里填入“factor(0x00a3474cbad937ee86ecba1f5fcd5c1433)”


D:\用户目录\下载\yafu-1.34\yafu-x64.exe
06/26/16 14:03:55 v1.34.5 @ USER-20160324FK, System/Build
Using GMP-ECM 6.3, Powered by GMP 5.1.1
detected Intel(R) Core(TM) i3 CPU M 370 @ 2.40GHz
detected L1 = 32768 bytes, L2 = 3145728 bytes, CL = 64 by
measured cpu frequency ~= 2387.486480
using 20 random witnesses for Rabin-Miller PRP checks
======= Welcome to YAFU (Yet Another Factoring Utility) =
======= bbuhrow@gmail.com =
======= Type help at any time, or quit to quit =======
cached 78498 primes. pmax = 999983
还原(R)
移动(M)
大小(S)
最小化(N)
最大化(X)
关闭(C)
编辑(E)
默认值(D)
属性(P)
标记(K)
复制(Y) Enter
粘贴(P)
全选(S)
滚动(L)
查找(F)...
应用程序
应用程序


D:\用户目录\下载\yafu-1.34\yafu-x64.exe
06/25/16 16:01:44 v1.34.5 @ USER-20160324FK, System/Build Info:
Using GMP-ECM 6.3, Powered by GMP 5.1.1
detected Intel(R) Core(TM) i3 CPU M 370 @ 2.40GHz
detected L1 = 32768 bytes, L2 = 3145728 bytes, CL = 64 bytes
measured cpu frequency ~= 2388.368540
using 20 random witnesses for Rabin-Miller PRP checks
======= Welcome to YAFU (Yet Another Factoring Utility) =======
======= bbuhrow@gmail.com =======
======= Type help at any time, or quit to quit =======
cached 78498 primes. pmax = 999983
>> factor(0x00a3474cbad937ee86ecba1f5fcd5c1433)

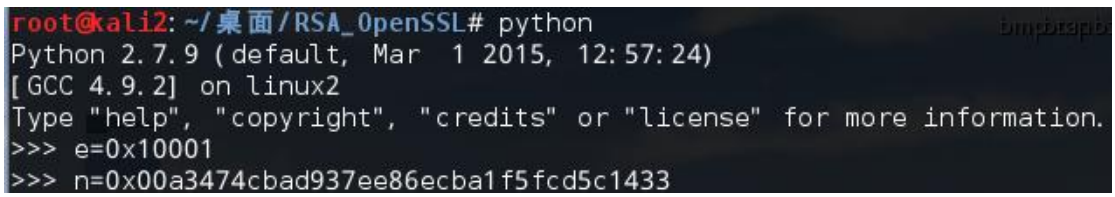


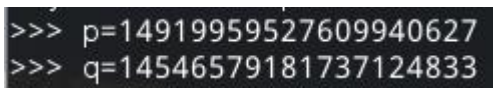
OKAY！酷！仅用了 0.2420 秒，就发现了模数 n 的质因子。

- P’=14919959527609940627
- q’=14546579181737124833

接下来将用到一些数学计算，让我们使用 Python：

```
>>> e=0x10001
>>> n=0x00a3474cbad937ee86ecba1f5fcd5c1433
>>> p=14919959527609940627
>>> q=14546579181737124833
```





接下来，我们主要按着前面的步骤 3-5 来破解出 d。让我们首先来计算 phi(n)：

```
>>> phi = (p-1)*(q-1)
```

接下来，我们借用扩展欧几里得算法（extended euclidean algorithm）来计算模逆：

```
>>> 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 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):
...     gcd, x, y = egcd(a, m)
...     if gcd !=1:
...             return None # modular inverse does not exist
...     else:
...             return x % m
...
>>>
```

```
>>> def modinv( a,  m):
...     gcd,  x,  y = egcd( a,  m)
...     if gcd !=1:
...             return None # modular inverse does not exist
...     else:
...             return x % m
...
```

```
>>> d=modinv(e,phi)
>>> d
157663448858086466466849277987508520897L
>>>hex(d)
```

```
>>> d=modinv( e, phi)
>>> d
157663448858086466466849277987508520897L
>>> hex( d)
' 0x769ce02fc886ea3e6efe74a6124907c1L'
>>>
```

如果详细地观察 d 的十六进制编码，你会发现它和我们前面的私钥值一致。

```
privateExponent:
    76:9c:e0:2f:c8:86:ea:3e:6e:fe:74:a6:12:49:07:
    c1
```

YAY，我们成功破解了私钥。

现在，我们还能做到手算密文，但是我更想生成一个私钥文件并借用 OpenSSL。这一做法并不直接使用 d 来解密，但是引入 3 个变量后将变得更高效。

```
>>> dp = d % p
>>> dq = d % q
```

```
>>> qi = pow(q, p - 2, p)
```

```
>>> dp = d % p
>>> dq = d % q
>>> qi = pow( q,  p  -  2,  p)
```

现在，我们可以使用 crypto.stackexchange.com 中的代码片段来生成 PEM 编码的私钥文件。你可能需要在系统中安装 python2-pyans1。

注：我到 https://sourceforge.net/projects/pyasn1/这个地方下载了 python2-pyans1。安装 python2-pyans1 时只要解压后，再到文件夹中执行“python setup.py install”命令。

```
root@kali2: ~/桌面/pyasn1-modules-0.0.8# python setup.py install
running install
running bdist_egg
running egg_info
writing requirements to pyasn1_modules.egg-info/requires.txt
writing pyasn1_modules.egg-info/PKG-INFO
writing top-level names to pyasn1_modules.egg-info/top_level.txt
writing dependency_links to pyasn1_modules.egg-info/dependency_links.txt
reading manifest file 'pyasn1_modules.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
```

破解得私钥的 Python 脚本如下：

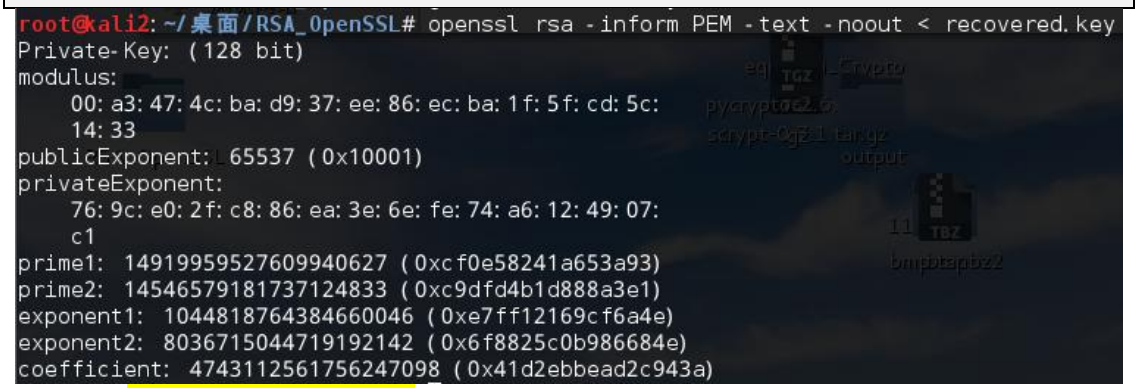
```
import pyasn1.codec.der.encoder
import pyasn1.type.univ
import base64

e=0x10001
n=0x00a3474cbad937ee86ecba1f5fcd5c1433
p=14919959527609940627
q=14546579181737124833
phi = (p-1)*(q-1)
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):
    gcd, x, y = egcd(a, m)
    if gcd !=1:
        return None # modular inverse does not exist
    else:
        return x % m

d=modinv(e,phi)
dp = d % p
dq = d % q
qi = pow(q, p - 2, p)

def pempriv(n, e, d, p, q, dP, dQ, qInv):
```

```
        template = '-----BEGIN RSA PRIVATE KEY-----\n{}-----END RSA PRIVATE KEY-----\n'
        seq = pyasn1.type.univ.Sequence()
        for x in [0, n, e, d, p, q, dP, dQ, qInv]:
            seq.setComponentByPosition(len(seq), pyasn1.type.univ.Integer(x))
        der = pyasn1.codec.der.encoder.encode(seq)
        return template.format(base64.encodestring(der).decode('ascii'))

key = pempriv(n,e,d,p,q,dp,dq,qi)
key

f = open("recovered.key","w")
f.write(key)
f.close()
```

OK!

让我们看看 OpenSSL 如何解析这把破解得的钥匙。

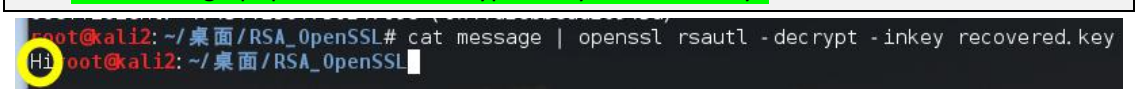
```
# openssl rsa -inform PEM -text -noout < recovered.key
Private-Key: (128 bit)
modulus:
    00:a3:47:4c:ba:d9:37:ee:86:ec:ba:1f:5f:cd:5c:
    14:33
publicExponent: 65537 (0x10001)
privateExponent:
    76:9c:e0:2f:c8:86:ea:3e:6e:fe:74:a6:12:49:07:
    c1
prime1: 14919595527609940627 (0xcf0e58241a653a93)
prime2: 14546579181737124833 (0xc9dfd4b1d888a3e1)
exponent1: 1044818764384660046 (0xe7ff12169cf6a4e)
exponent2: 8036715044719192142 (0x6f8825c0b986684e)
coefficient: 4743112561756247098 (0x41d2ebbead2c943a)
```



注：exponent1 和 exponent1 有区别，但是这并没有关系。

最后一步是使用破解的秘钥进行解密：

```
# cat message | openssl rsautl -decrypt -inkey recovered.key
```



YAY, we did it! :)

结论

我希望您能感受到非对称加密系统是 COOL 的，但是在使用它的时候如果使用的是 small keys 的话，则一点也不。

当秘钥的位数达到 300 时，要进行质数分解的时间对于一台个人电脑来说，将达到 1 个小时。甚至有一个 RSA factoring challenge，它能攻破的最大模数的位数达到了 768 位。这是一个真正的问题，但是人们在 GitHub 上用的就是这么短的 small ssh keys。有时候，太小的公钥会导致严重的安全问题。

我们推荐使用的秘钥程度是 2048 位，即使对于智能手机，这也几乎不会造成性能损失。

代码

以下是完整的 Python 代码：

```
#!/usr/bin/python2
import pyasn1.codec.der.encoder
import pyasn1.type.univ
import base64
def recover_key(p, q, e, output_file):
    """Recoveres a RSA private key from:
    open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com
    p: Prime p
    q: Prime q
    e: Public exponent
    output_file: File to write PEM-encoded private key to"""
    # SRC: https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Extended_Euclidean_algorithm

def egcd(a, b):
    x,y,u,v = 0,1,1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
    return gcd, x, y

def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None # modular inverse does not exist
    else:
        return x % m
```

```
# SRC: http://crypto.stackexchange.com/questions/25498/how-to-create-a-pem-file-for-storing-an-rsa-key/25499#25499
def pempriv(n, e, d, p, q, dP, dQ, qInv):
    template = '-----BEGIN RSA PRIVATE KEY-----\n{}-----END RSA PRIVATE KEY-----\n'
    seq = pyasn1.type.univ.Sequence()
    for x in [0, n, e, d, p, q, dP, dQ, qInv]:
        seq.setComponentByPosition(len(seq), pyasn1.type.univ.Integer(x))
    der = pyasn1.codec.der.encoder.encode(seq)
    return template.format(base64.encodestring(der).decode('ascii'))
e=0x10001
n=0x00a3474cbad937ee86ecba1f5fcd5c1433
p=14919959527609940627
q=14546579181737124833
phi = (p-1)*(q-1)
n = p * q
phi = (p -1)*(q-1)
d = modinv(e, phi)
dp = d % p
dq = d % q
qi = pow(q, p - 2, p)
key = pempriv(n, e, d, p, q, dp, dq, qi)
f = open('output_file',"w")
f.write(key)
f.close()
```