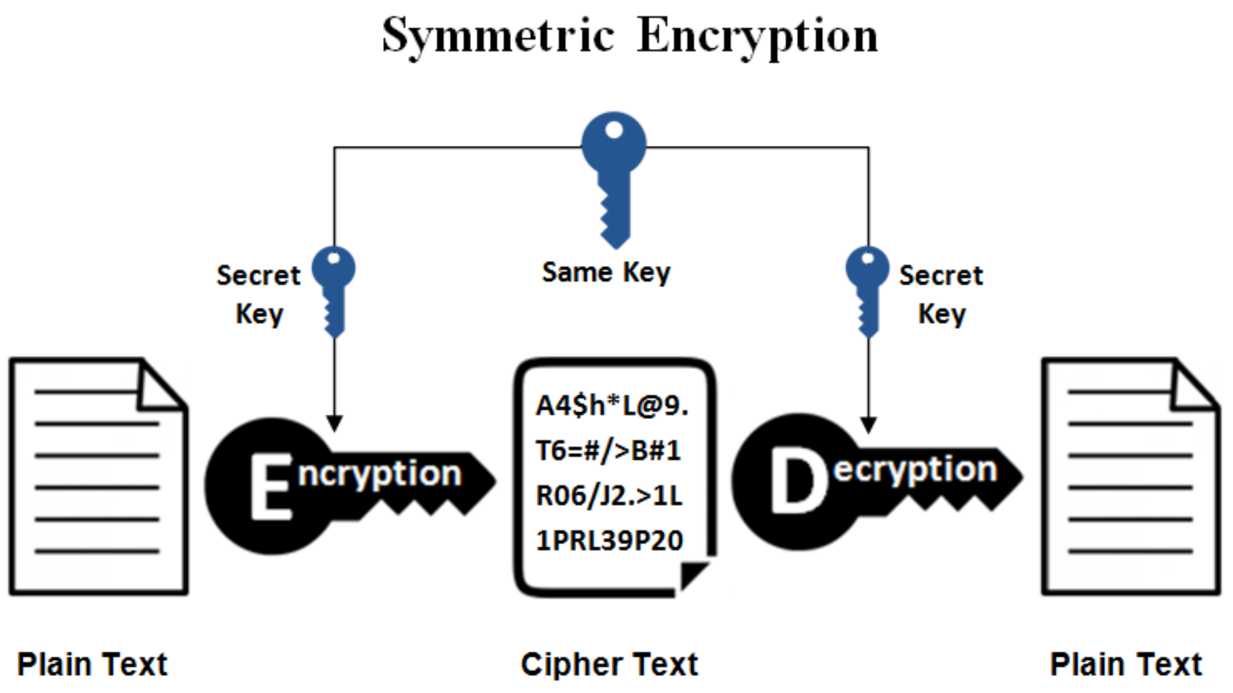
CTF-Crypto

2020/09/18

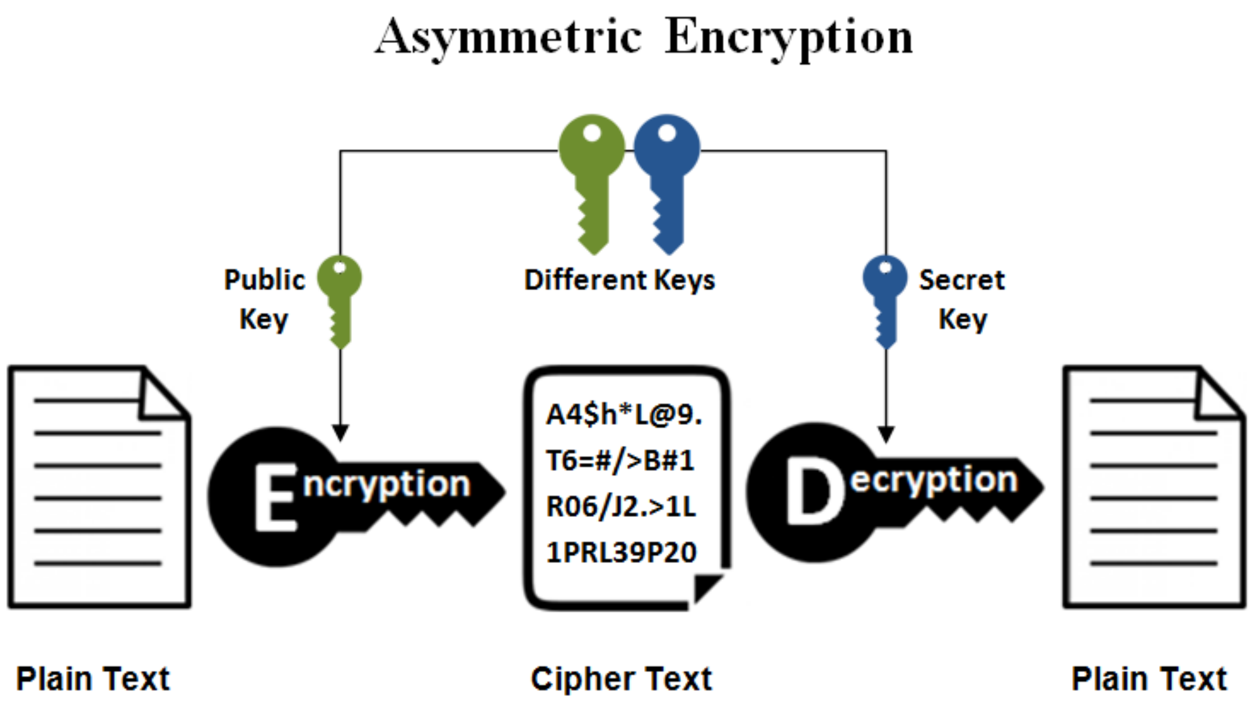
目录 / Contents

一、RSA简介

对称密码

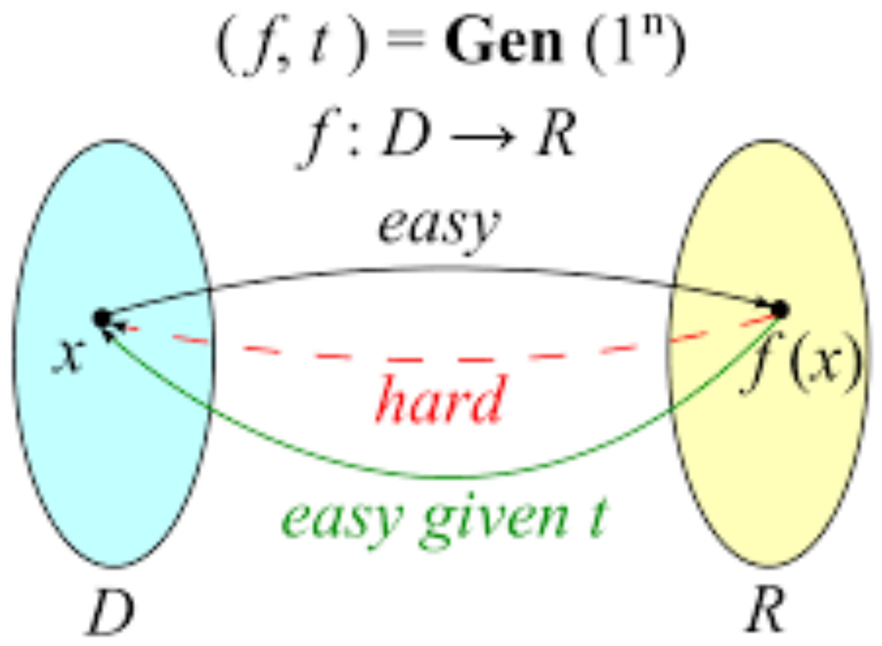


非对称密码



一、RSA简介

单向函数

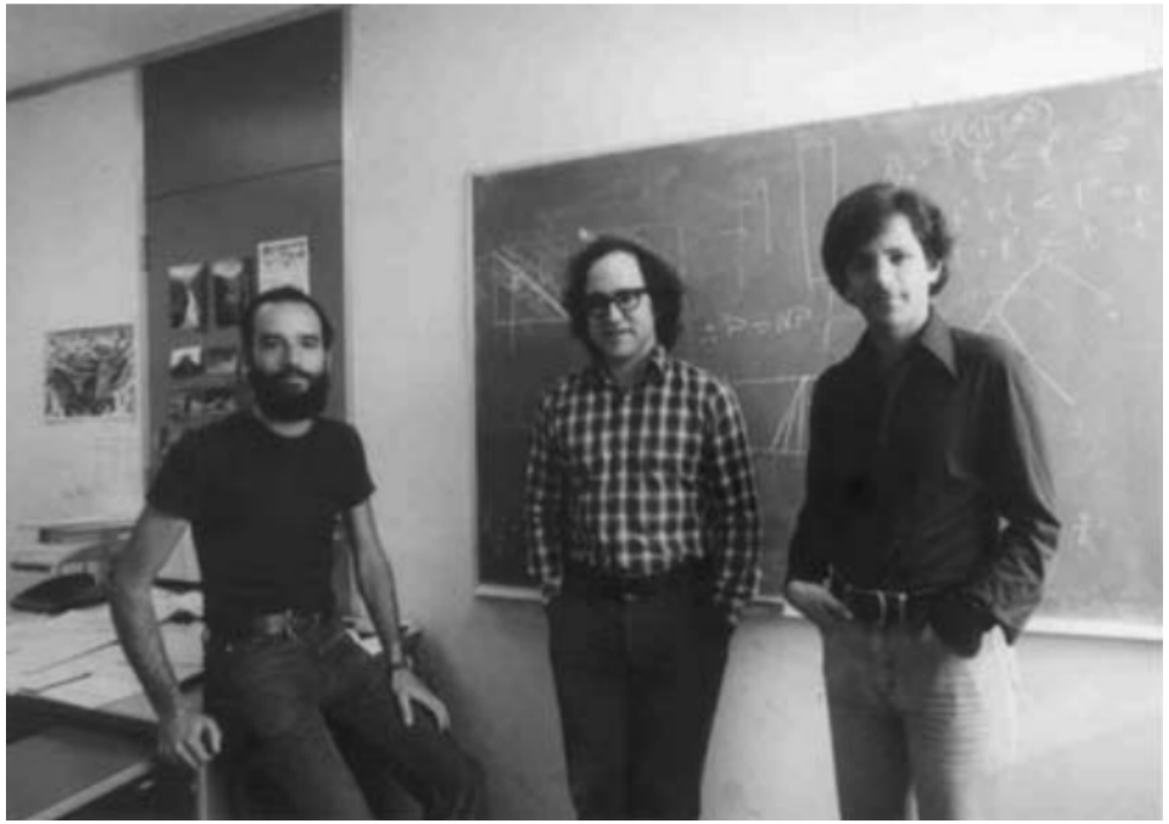


- 对每一个输入x，函数值f(x)都很容易计算
- 对随机给出的函数值f(x)，算出原始输入却比较困难x
- 使用陷门信息(trapdoor information)则可以反逆

一、RSA简介

RSA起源

RSA是1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）一起提出的。当时他们三人都在麻省理工学院工作。RSA就是他们三人姓氏开头字母拼在一起组成的。



一、RSA简介

RSA原理

公钥与私钥的产生

1. 选择两个不同的大素数p和q，计算N=p·q。
2. 计算phi(n) = (p – 1)·(q – 1)。
3. 选择一个小于phi(n)的整数e，使得e与phi(n)互素。
4. 计算e关于phi(n)的模逆元d。
5. 将p和q的记录销毁。

将(N, e)作为公钥，将(N, d)作为私钥。

一、RSA简介

RSA原理

加密消息

mod $\phi(N)$

$$c \equiv m^e \ (mod\ N)$$

解密消息

mod $\phi(N)$

$$m \equiv c^d \ (mod\ N)$$

$$m \equiv c^d \equiv (m^e)^d \equiv m^{ed\ (mod\ \phi(N))} (mod\ N)$$

二、RSA模数相关攻击

分解模数n

In January 2002, Franke et al. announced the factorisation of a 158-digit cofactor of $2^{953}+1$, using a couple of months on about 25 PCs at the University of Bonn, with the final stages done using a cluster of six Pentium-III PCs.

In April 2003, the same team factored RSA-160 using about a hundred CPUs at BSI, with the final stages of the calculation done using 25 processors of an SGI Origin supercomputer.

The 576-bit RSA-576 was factored by Franke, Kleinjung and members of the NFSNET collaboration in December 2003, using resources at BSI and the University of Bonn; soon afterwards, Aoki, Kida, Shimoyama, Sonoda and Ueda announced that they had factored a 164-digit cofactor of $2^{1826}+1$.

A 176-digit cofactor of $11^{281}+1$ was factored by Aoki, Kida, Shimoyama and Ueda between February and May 2005 using machines at NTT and Rikkyo University in Japan.[1]

The 663-bit RSA-200 challenge number was factored by Franke, Kleinjung et al. between December 2003 and May 2005, using a cluster of 80 Opteron processors at BSI in Germany; the announcement was made on 9 May 2005.[2] They later (November 2005) factored the slightly smaller RSA-640 challenge number.

On December 12, 2009, a team including researchers from the CWI, the EPFL, INRIA and NTT in addition to the authors of the previous record factored RSA-768, a 232-digit semiprime.[3] They used the equivalent of almost 2000 years of computing on a single core 2.2 GHz AMD Opteron.

In November 2019, the 795-bit RSA-240 was factored by Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé and Paul Zimmermann.[4][5]

In February 2020, the factorization of the 829-bit RSA-250 was completed.[6]

二、RSA模数相关攻击

分解模数n

(e,n)=(0x10001,0x291733BAB061EF9C599139CB3E40A5C762B6F448FFFFFFFFFFFFFF)

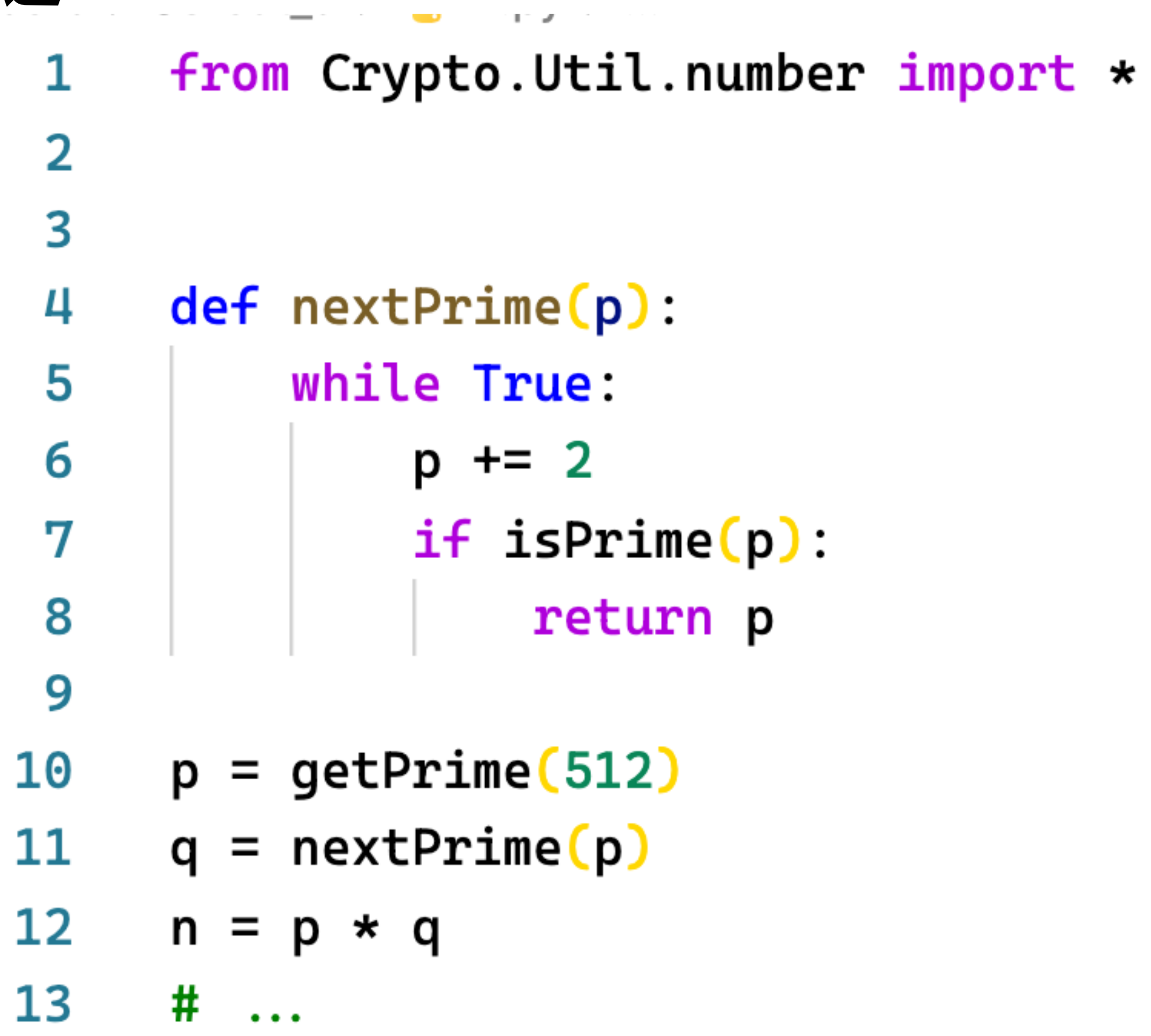
c=237200C0F72B97DB55BA37C7AACBB61A26A0CB47D294726259C4DF

http://factordb.com/

https://www.alpertron.com.ar/ECM.HTM

二、RSA模数相关攻击

p、q相近



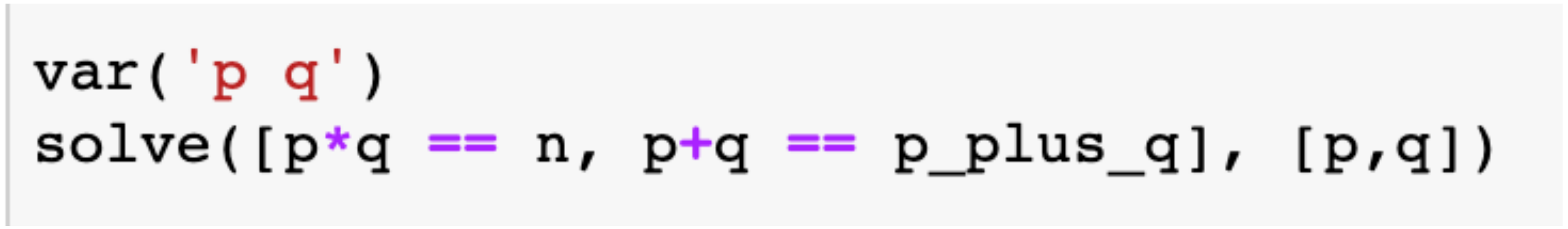
```python
from Crypto.Util.number import *


def nextPrime(p):
    while True:
        p += 2
        if isPrime(p):
            return p

p = getPrime(512)
q = nextPrime(p)
n = p * q
# ...
```

二、RSA模数相关攻击

已知p+q或p-q

$$p \cdot q = n$$
$$p + q = a$$

解一个二元二次方程组


```
var('p q')
solve([p*q == n, p+q == p_plus_q], [p,q])
```

二、RSA模数相关攻击

多素数的模数

三素数乘积

$$n = p \cdot q \cdot r$$

$$\phi(n) = (p-1) * (q-1) * (r-1)$$

欧拉函数

若 $n = \prod_{p|n, 1 \le p < n} p^{a_p}$

则 $\varphi(n) = \prod (p-1) p^{a_p - 1}$

素数幂

$$n = p^r$$

$$\phi(n) = p^r - p^{r-1}$$

二、RSA模数相关攻击

模不互素

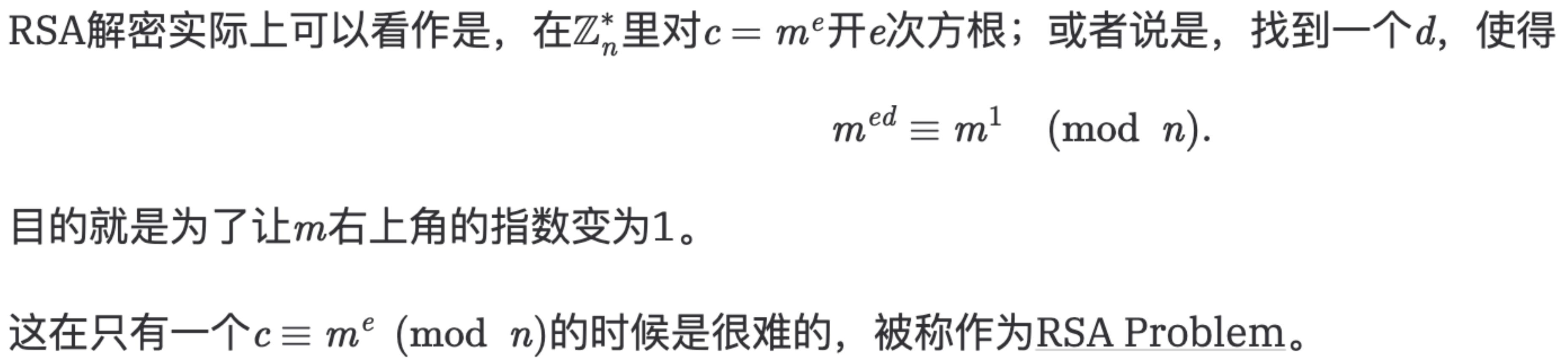
```
p = getPrime(1024)
q1 = getPrime(1024)
q2 = getPrime(1024)

n1 = p * q1
n2 = p * q2
```

二、RSA模数相关攻击

共模攻击

攻击条件

当两个用户使用相同的模数 N、不同的私钥时，加密同一明文消息时即存在共模攻击。



RSA解密实际上可以看作是，在$\mathbb{Z}_n^*$里对$c = m^e$开e次方根；或者说是，找到一个d，使得

$$m^{ed} \equiv m^1 \pmod{n}.$$

目的就是为了让m右上角的指数变为1。

这在只有一个$c \equiv m^e \pmod{n}$的时候是很难的，被称作为RSA Problem。

二、RSA模数相关攻击

共模攻击

但是现在有两组这样的关系：

$$\begin{aligned} m^{e_1} &\equiv c_1 \pmod{n} \\ m^{e_2} &\equiv c_2 \pmod{n}. \end{aligned}$$

我们可以通过Extended Euclidean algorithm来计算出

$$re_1 + se_2 = 1. \tag{1}$$

这样就能把指数给变为1：

$$c_1^r c_2^s \equiv m^{re_1+se_2} \equiv m^1 \pmod{n}.$$

但是r和s中必有一个是负的，不然等式(1)左边全都是正的，但等式右边是1，不可能。

模运算里可没有倒数，但可以用逆元来处理。

（without loss of generality）假设s是负的，令s^+表示$|s| = -s$，那么有：

$$m^1 \equiv m^{re_1+se_2} \equiv (m^{e_1})^r + (m^{e_2})^{-s^+} \equiv c_1^r \cdot (c_2^{-1})^{s+} \pmod{n}$$

二、RSA模数相关攻击

广播攻击

攻击条件

如果一个用户使用同一个加密指数 e 加密了同一个密文，并发送给了其他 e 个用户。那么就会产生广播攻击。这一攻击由 Håstad 提出。

攻击原理

这里我们假设 e 为 3，并且加密者使用了三个不同的模数 n_1, n_2, n_3 给三个不同的用户发送了加密后的消息 m，如下

$$
\begin{aligned}
c_1 &= m^3 \bmod n_1 \\
c_2 &= m^3 \bmod n_2 \\
c_3 &= m^3 \bmod n_3
\end{aligned}
$$

二、RSA模数相关攻击

广播攻击

```
[+]Generating challenge 4
[+]e=3
[+]m=random.getrandbits(512)
[+]n1=0x1ec2150f6e573adba01b2fe569ae7a0a2d02d82d788c6571ddfdb411af18666e7b64c47defa9e292682ad4e5b07d690e372cf5baad656fd2227
01d8acc68bf35646a4343c5aa88de2e8859abac9884a72c7a4525b813644f8f806465feb0a03b6d734995a7ed5a751a49e35d1c4bac592aefd91dee81f
1fa9ac027fc3647d4ebL
[+]c1=pow(m,e,n1)=0xb081a5dbd2ab11925407875e217aa98754e944c4fda52a3341d1cb0bd4c6621a64757e119601918665c9877a33241c3d2483c
00cf822dacf257617b6f0dc8de05d4b59ed5958f52dfce50b014d900ffe4d9e375824bc648adb72ecc4c4ecdc9f3be49fced7424d0ed696b6e98b1f6ddea
a79ebca0a592ba05bc11f98aa9ab1cL
[+]n2=0x1ca49ec4a77cfcfcbbf393e9772538c2adf63d0649226bb2aa357178c0a56f6481c39b7e96da90750bd73b067e8b52e2133bdba9f9bde8d868c
d682826c2ee10a7a2958b887b07df1e05e38b515da13c4346b948831af253744e2b7cc90a9414fa5b4ada0327236ae29a7b010d8d9e6529491566e7fb
71c91746e43bbd6ebe8fL
[+]c2=pow(m,e,n2)=0xf570f81b5fb68810dce6811a3a6c86c507e250ac903f412cd89bfc572652b9376b03105e410754422583d9a6522f607a9bcceb14
357688a5b1eeafc87066b6304872091ff1760ad6a9d8d72d4cb64b51b559cbb8c7d790303d9fa491fc3f7d6e6bde370cff2c89528978fbfaf2724e63b334
7e3cc0129e1b79056c0e9653deL
[+]n3=0x1f3497868702f5500fc66239f280303bb2129f10c3607ff4aca342ecdb1850bbaf9404b0e7533e6a6d0bdc71bb3336393da5bed3c6f7ab8c4e63
b9e37c05a09a3c91269c3385b19759f36b9b1ebbcc4245a1c46ddedcbe80865701942e38cedc82b54630659772e8de8b33064fad6d5551c2e19ed8fa2
0541d2ca3818d5bc6e1L
[+]c3=pow(m,e,n3)=0x480830044351b6d4f86b9968e56a5a3b18b1f966851229f3a500f870d8a3ad364944c18701d67cf02f876a5ec353935ee4d3d7e
313f0db0867da70a40458577764540ef60446c7a71577598498b89f2d706013936c9eb9b0f730a27d197dc64370a1e772fcca8ae59a56a0de0dbcbd0d
92228df2efd3fb64dcf87a27e842c1eL
[-]long_to_bytes(m).encode('hex')=
```

三、RSA指数相关攻击

小公钥指数攻击

攻击条件

e 特别小，比如 e 为 3。

攻击原理 ¶

假设用户使用的密钥 $e=3$。考虑到加密关系满足：

$$c \equiv m^3 \bmod N$$

则：

$$\begin{aligned} m^3 &= c + k \times N \\ m &= \sqrt[3]{c + k \times n} \end{aligned}$$

攻击者可以从小到大枚举 k，依次开三次根，直到开出整数为止。

三、RSA指数相关攻击

已知e，d分解n

$$ed \equiv 1 \pmod{\phi(n)}$$
$$ed = 1 + k \cdot \phi(n), k < e$$

穷举k，计算出$\phi(N)$

$$\phi(n) = (p-1)(q-1) = n - (p+q) + 1$$
$$n = p \cdot q$$

解一个二元二次方程组

三、RSA指数相关攻击

Wiener Attack

攻击条件

在 d 比较小（$d < \frac{1}{3}N^{\frac{1}{4}}$）时，攻击者可以使用 **Wiener's Attack** 来获得私钥。

$$\phi(n) = n - (p+q) + 1$$
$$ed = 1 + k\phi(n)$$

$$ed = 1 + k(pq - (p+q) + 1)$$
$$\frac{e}{n} = \frac{k}{d}(1-\delta)$$

https://github.com/pablocelayes/rsa-wiener-attack

四、RSA LSB Oracle Attack

LSB Oracle Attack

假设目前存在一个 Oracle，它会对一个给定的密文进行解密，并且会检查解密的明文的奇偶性，并根据奇偶性返回相应的值，比如 1 表示奇数，0 表示偶数。那么给定一个加密后的密文，我们只需要 log(N) 次就可以知道这个密文对应的明文消息。

```python
cc = int(raw_input('Your encrypted message:').strip())
mm = k.decrypt(cc)
if mm & 1 == 1:
    print 'The plain of your decrypted message is odd!'
else:
    print 'The plain of your decrypted message is even!'
```

四、RSA LSB Oracle Attack

LSB Oracle Attack

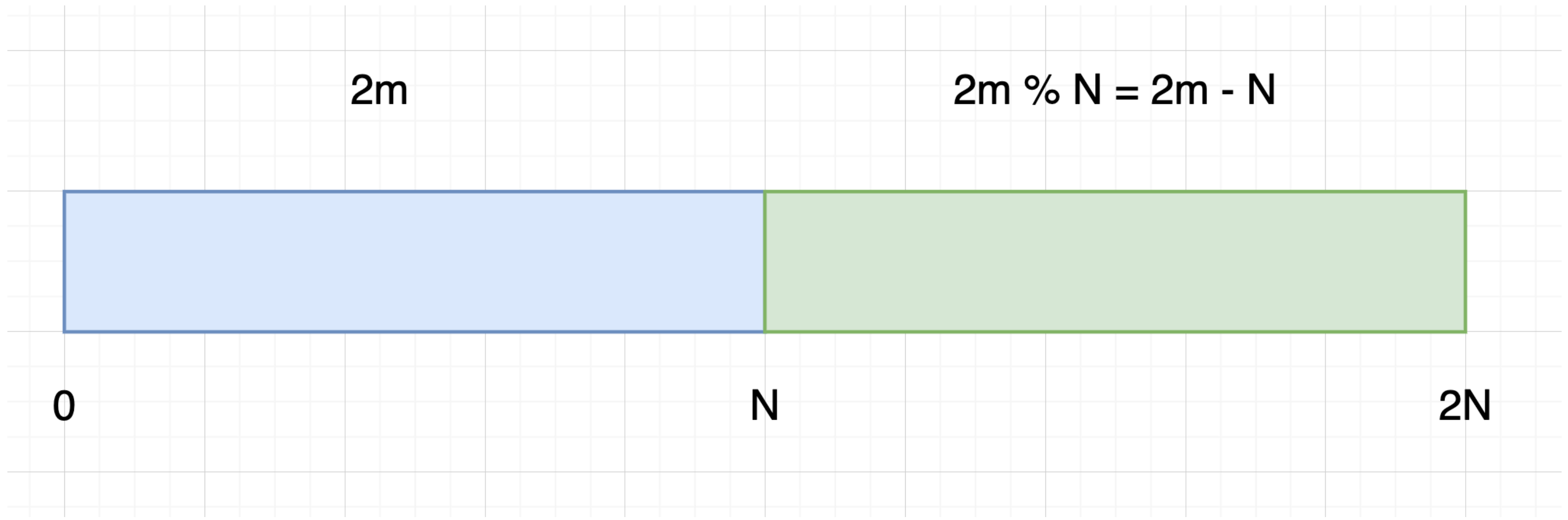
$$C = P^e \bmod N$$

第一次时，我们可以给服务器发送

$$C * 2^e = (2P)^e \bmod N$$

服务器会计算得到

$$2P \bmod N$$



- 服务器返回奇数，即 $2P \bmod N$ 为奇数，则说明 2P 大于 N，且减去了奇数个 N，又因为 $2P < 2N$，因此减去了一个 N， 即 $\frac{N}{2} \leq P < N$，我们还可以考虑向下取整。
- 服务器返回偶数，则说明 2P 小于 N。即 $0 \leq P < \frac{N}{2}$，我们还可以向下取整。

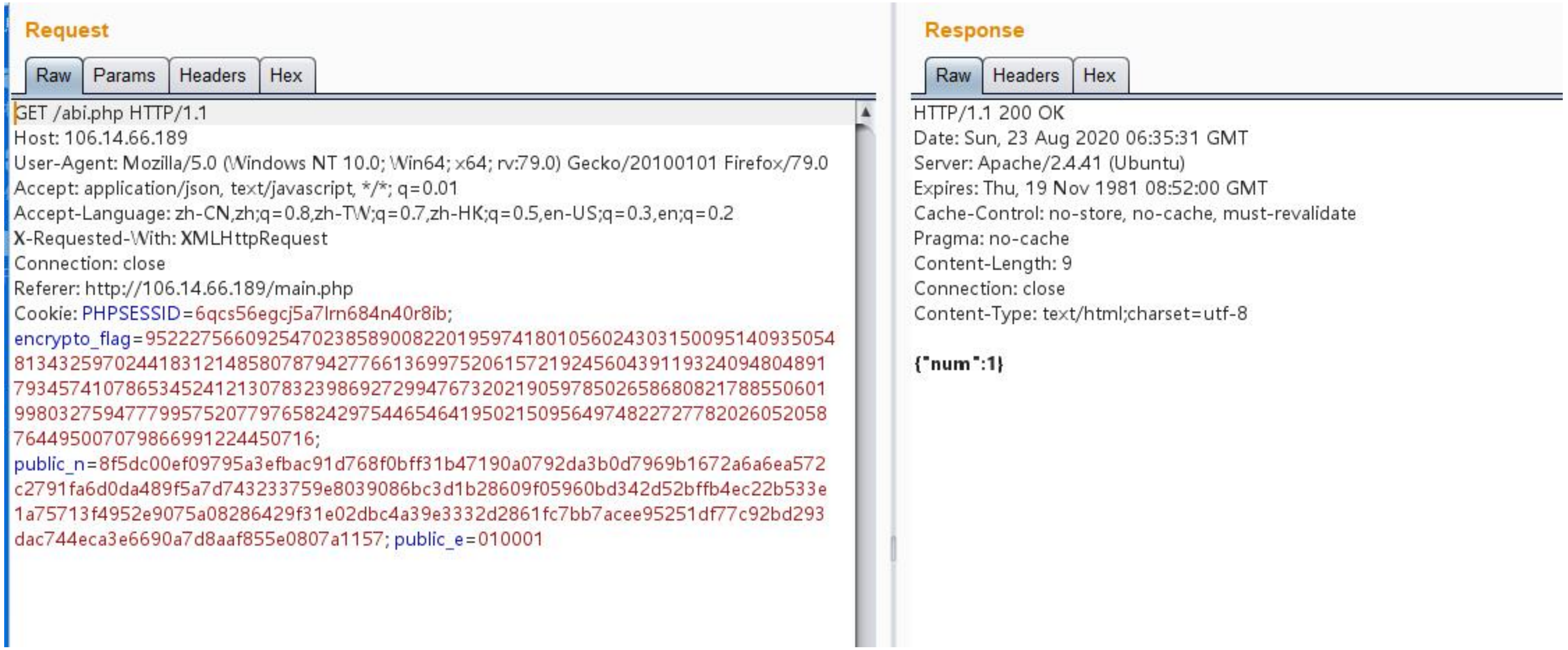
四、RSA LSB Oracle Attack

LSB Oracle Attack

```php
<?php
session_start();
header("Content-type:text/html;charset=utf-8");

        $data = json_decode($json_string, true);

        $rand_number = isset($_POST['this_is.able']) ? $_POST['this_is.able'] : mt_rand();
        $n = gmp_init($data['n']);
        $d = gmp_init($data['d']);
        $c = gmp_init($rand_number);
        $m = gmp_powm($c,$d,$n);
        $v3 = gmp_init('3');
        $r = gmp_mod($m,$v3);
        $result=(int)gmp_strval($r);
        $dice = array("num"=>$result);
        $json_obj = json_encode($dice);
        echo $json_obj;
```

四、RSA LSB Oracle Attack

LSB Oracle Attack


Request

Raw | Params | Headers | Hex

```
GET /abi.php HTTP/1.1
Host: 106.14.66.189
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
X-Requested-With: XMLHttpRequest
Connection: close
Referer: http://106.14.66.189/main.php
Cookie: PHPSESSID=6qcs56egcj5a7lrn684n40r8ib;
encrypto_flag=95222756609254702385890082201959741801056024303150095140935054
81343259702441831214858078794277661369975206157219245604391193240948048917
9345741078653452412130783239869272994767320219059785026586808217885506019
9803275947779957520779765824297544654641950215095649748227277820260520587
644950070798669912244507I6;
public_n=8f5dc00ef09795a3efbac91d768f0bff31b47190a0792da3b0d7969b1672a6a6ea572
c2791fa6d0da489f5a7d743233759e8039086bc3d1b28609f05960bd342d52bffb4ec22b533e
1a75713f4952e9075a08286429f31e02dbc4a39e3332d2861fc7bb7acee95251df77c92bd293
dac744eca3e6690a7d8aaf855e0807a1157; public_e=010001
```

Response

Raw | Headers | Hex

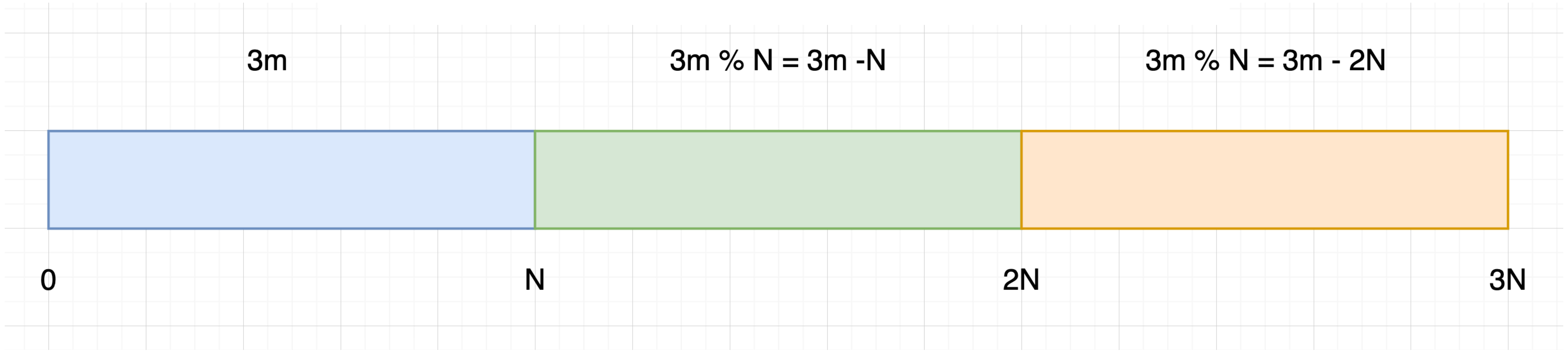
```
HTTP/1.1 200 OK
Date: Sun, 23 Aug 2020 06:35:31 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 9
Connection: close
Content-Type: text/html;charset=utf-8

{"num":1}
```

四、RSA LSB Oracle Attack

LSB Oracle Attack

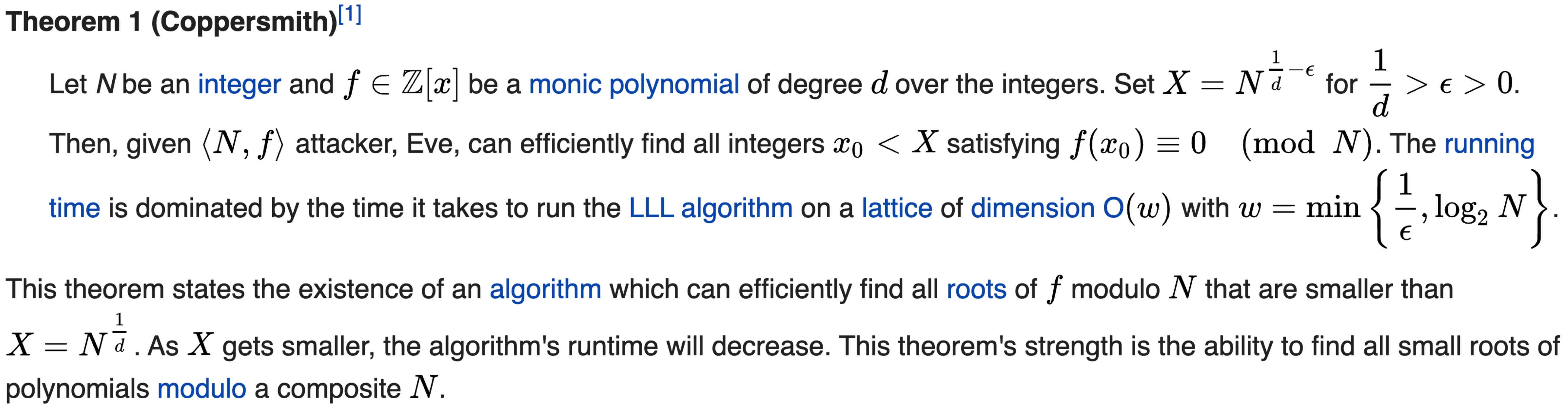
$$C \cdot 3^e \pmod{n}$$

$$(C \cdot 3^e)^d = m \cdot 3 \pmod{n}$$



五、RSA coppersmith相关攻击

Coppersmith定理



Theorem 1 (Coppersmith)[1]

Let N be an integer and $f \in \mathbb{Z}[x]$ be a monic polynomial of degree d over the integers. Set $X = N^{\frac{1}{d}-\epsilon}$ for $\frac{1}{d} > \epsilon > 0$. Then, given $\langle N, f \rangle$ attacker, Eve, can efficiently find all integers $x_0 < X$ satisfying $f(x_0) \equiv 0 \pmod{N}$. The running time is dominated by the time it takes to run the LLL algorithm on a lattice of dimension $O(w)$ with $w = \min\left\{\frac{1}{\epsilon}, \log_2 N\right\}$.

This theorem states the existence of an algorithm which can efficiently find all roots of f modulo N that are smaller than $X = N^{\frac{1}{d}}$. As X gets smaller, the algorithm's runtime will decrease. This theorem's strength is the ability to find all small roots of polynomials modulo a composite N.

五、RSA coppersmith相关攻击

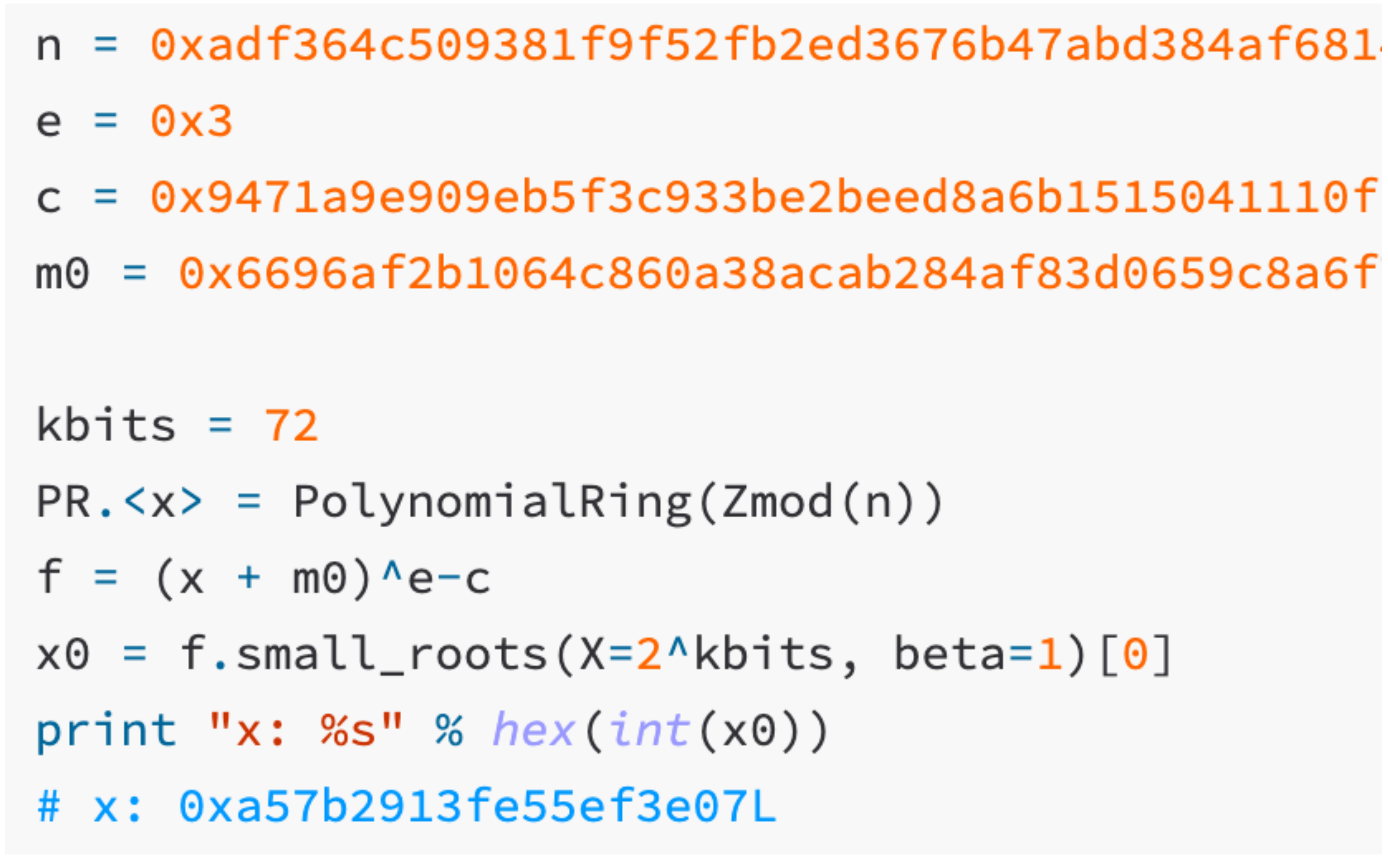
已知m的高位

```
[+]Generating challenge 1
[+]n=0xadf364c509381f9f52fb2ed3676b47abd384af6814cb30c3480f562470eb6b1e30a93cf9493e98587a97b05725a3dd7
af7a0a906bd1583e8ced2d1457954fb250b827002e148e8c58f7414f4351c51c62d538f1c10c0404c98d103db69dfdb02c535
4871b179f854fcc4d2ec8d83855c764fa766578617888a6ec2668260fca3L
[+]e=3
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x9471a9e909eb5f3c933be2beed8a6b1515041110fca47701e64fa36adb8748a10ba939571e7904849f4c
0666c5aed8cf7d8c4978cc5e18f564fe0bb0311e22b4a04c5ccae6603bbb65adaa9668d9ca6fc479960bb94546eaa1de75877
ce1c40262d21894e966a4436128d9edf49d72f71df1d5c77ee0dc976e97c5740f07828dL
[+]((m>>72)<<72)=0x6696af2b1064c860a38acab284af83d0659c8a6f7aca6e147ecb5874a47108074608c619b5f001b0355
8da7e0c4546e3c8318ef70e287800000000000000000000L
[-]long_to_bytes(m).encode('hex')=
```

五、RSA coppersmith相关攻击

已知m的高位

$$f = (m_0 + \delta)^e - c \pmod{N}$$



```
n = 0xadf364c509381f9f52fb2ed3676b47abd384af681
e = 0x3
c = 0x9471a9e909eb5f3c933be2beed8a6b151504111 0f
m0 = 0x6696af2b1064c860a38acab284af83d0659c8a6f

kbits = 72
PR.<x> = PolynomialRing(Zmod(n))
f = (x + m0)^e-c
x0 = f.small_roots(X=2^kbits, beta=1)[0]
print "x: %s" % hex(int(x0))
# x: 0xa57b2913fe55ef3e07L
```

五、RSA coppersmith相关攻击

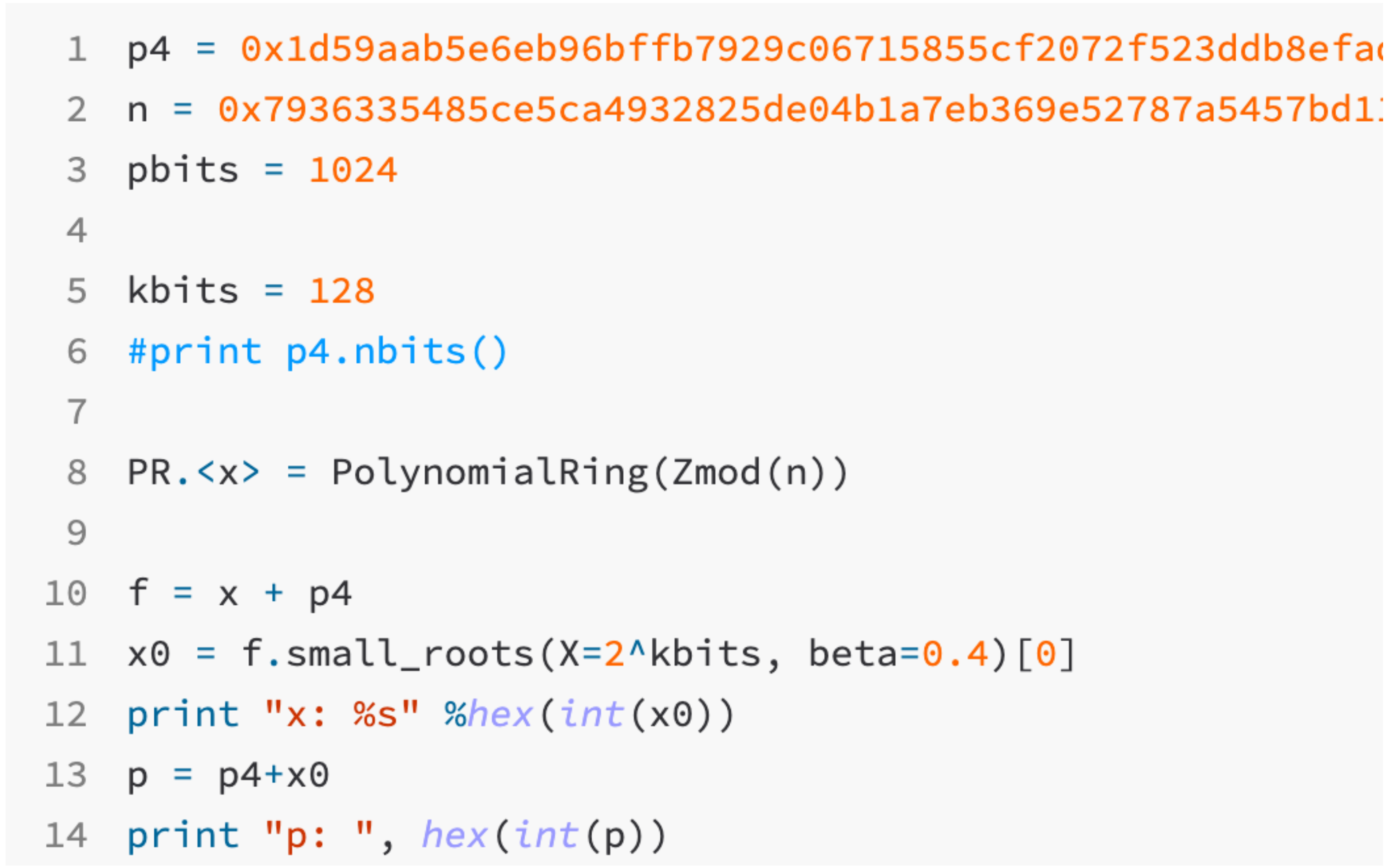
已知p的高位

```
[+]Generating challenge 2
[+]n=0x7936335485ce5ca4932825de04b1a7eb369e52787a5457bd115e5fc0639fd9df1e27ddb527a69c08ee4c52c3e457af
a91277cb1af71c281e99858acc62b77075072036f58f0a0bb40f5ab3462a4f18873c3c681304153a8c17caac65682c34cc752d
81b758091e457f1ae5f0759995c341e099089297212de519363c59c5cbL
[+]e=65537
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x33e3d895b445ed22acc7ed9e771f27bc5314a671706ea95996a9f1ae8e9f1cc1e18effd0178d4953c30d
9adb242aac8474fc666161c7fa12bcd2738d65435190882f0f7432fb5b57dddb94e7e047e499503921a1e9a5d664c03a7be7
70675b8482a65f63ba18c2d300c11c0a46d8d11334df50780af78d90d0b0eeba3f3c19L
[+]((p>>128)<<128)=0x1d59aab5e6eb96bffb7929c06715855cf2072f523ddb8efadc57d2707638a87ab3c68304b9aadd1b2f
a897628eb73ea100000000000000000000000000000000L
[-]long_to_bytes(m).encode('hex')=
```

五、RSA coppersmith相关攻击

已知p的高位

$$f = p_0 + \delta \pmod{p}$$



```
p4 = 0x1d59aab5e6eb96bffb7929c06715855cf2072f523ddb8efa
n = 0x7936335485ce5ca4932825de04b1a7eb369e52787a5457bd1
pbits = 1024

kbits = 128
#print p4.nbits()

PR.<x> = PolynomialRing(Zmod(n))

f = x + p4
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]
print "x: %s" %hex(int(x0))
p = p4+x0
print "p: ", hex(int(p))
```