CTF中常见的RSA相关问题总结

前言

关于RSA安全性的讨论在网络上已经有很多很好的资料，关于RSA在CTF领域的题目的解答和总结也不乏精彩的文章，所以本文并没有什么耀眼的内容，甚至全面也做不到。现在回头看觉得这些知识虽然要严谨证明不容易，但直观理解和运用起来都非常简单，以至于有些不好意思拿出来。但转念一想，闻道有先后，也许后来者能从中受益，也就斗胆放出来了。如果文中有任何不正确或不恰当的地方请留言指出。

理解基本概念后，代码就可以说明一切，所以本文将每种攻击方式的实现方法都提炼成了一个函数，在理解原理时会有帮助，在需要时也可以直接调用。

基础

RSA概要

在开始前可以通过 《RSA算法详解》 这篇文章了解关于RSA的基础知识，包括加解密方法，算法原理和可行性证明等。

应用流程

1. 选取两个较大的互不相等的质数p和q，计算`n = p * q`。
2. 计算`phi = (p-1) * (q-1)`。
3. 选取任意e，使得e满足`1<e<phi`且`gcd(e , phi) == 1`。
4. 计算e关于n的模逆元d， 即d满足`(e * d)% n ==1` 。
5. 加解密：`c = (m ^ e) % n`， `m = (c ^ d) % n`。其中m为明文，c为密文，(n,e)为公钥对，d为私钥，要求`0 <= m < n`。

理解模逆运算

- 如果`(a*b)%c==1`，那么a和b互为对方模c的模逆元/数论倒数，也写作

  $$a^{-1} \equiv b \pmod{c} \text{ 或 } ab \equiv 1 \pmod{c}$$ 。

- 关于最大公约数有一个基本事实：`给予两整数a、c，必存在整数x、y使得ax + cy = gcd(a,c)`，基于这个事实，当a,c互素即`gcd(a,c)==1`时，有`ax+cy=1`，那么就有`(a*x)%c==1`，所以x就是a 对c的模逆元。因此，a对c存在模逆元b的充要条件是`gcd(a,c)==1`。显然对于每一组`a,c`，存在一族满足条件的x，在求模逆元时我们取得是最小正整数解`x mod n`。
- 上述的基本事实很容易理解，因为a和c的最大公约数是gcd(a,b)，所以a和c都可表示为gcd(a,b)的整数倍，那么a和b的任意整系数的线性组合ax+by也必定能表示成gcd(a,c)的整数倍，他们当中最小的正整数就应该是gcd(a,c)。实际上最大公约数有一个定义就是：`a和b的最大公约数g是a和b的线性和中的最小正整数`。
- 求模逆元主要基于扩展欧几里得算法，贴一个Python实现：

```python
def egcd ( a , b ):
    if (b == 0):
        return 1, 0, a
    else:
        x , y , q = egcd( b , a % b ) # q = GCD(a, b) = GCD(b, a%b)
        x , y = y, ( x - (a // b) * y )
        return x, y, q
def mod_inv(a,b):
    return egcd(a,b)[0]%b #求a模b得逆元
```

- 求模逆也可直接利用gmpy2库。如 `import gmpy2;print gmpy2.invert(47,30)` 可求得47模30的逆为23。

模意义下的运算法则

```
(a + b) % n ≡ (a % n + b % n) % n
(a - b) % n ≡ (a % n - b % n) % n
(a * b) % n ≡ (a % n * b % n) % n
(a ^ b) % n ≡ ((a % n) ^ b) % n //幂运算

若 a ≡ b(mod n) ,则
1.对于任意正整数c,有a^c ≡ b^c(mod n)
2.对于任意整数c,有ac ≡ bc(mod n),a+c ≡ b+c(mod n),
3.若 c ≡ d(mod n),则a-c ≡ b-d(mod n),a+c ≡ b+d(mod n),ac ≡ bd(mod n)

如果ac≡bc (mod m)，且c和m互质，则a≡b (mod m) 。
[理解：当且仅当c和m互质,c^-1存在,等式左右可同乘模逆。]

除法规则：
在模n意义下，a/b不再仅仅代表这两个数相除，而是指 a+k1*n 和 b+k2*n这两个组数中任意两个相除，使商为整数
因此也就可以理解，除以一个数等价于乘以它的逆
a/b ≡ c(mod n) <=> a ≡ c*(b^-1) (mod n)，其中b模n的逆记作b的负一次方。

费马小定理：
a是整数,p是质数,则a^p==a(mod p),如果a不是p的倍数,还有a^(p-1) ≡ 1(mod p)
```

推荐文章 模运算总结 和 取模运算涉及的算法 。

欧几里得算法

欧几里得算法是求最大公约数的算法, 也就是中学学的 辗转相除法 。记 `gcd(a,b)` 为a和b的最大公约数，欧几里得算法的基本原理是 `gcd(a,b)==gcd(b,a%b),(b!=0)` 和 `gcd(a,0)==a` 。

Python实现如下：

```python
# 递归版
def gcd(a, b):
    return a if not b else gcd(b, a % b)

# 迭代版
def gcd2(a, b):
    while not b:
        a, b = b, a % b
        return a
```

扩展欧几里得算法

扩展欧几里得算法基于欧几里得算法，能够求出使得 `ax+by=gcd(a,b)` 的一组x,y。

这篇文章 解释得很到位，对照下图和以下递归版实现容易理解。

那么，假设我们知道了方程 $bx+(a \bmod b)y=\gcd(b,a \bmod b)$ 的一组整数解 (x',y')。

由于 $\gcd(a,b)=\gcd(b,a \bmod b)$，我们可以将上面两个方程联立起来，可以得到

$$ax+by=bx'+(a \bmod b)y'$$

如果我们用 $a \bmod b=a-\lfloor \frac{a}{b} \rfloor b$ 来替换

$$ax+by=bx'+(a-\lfloor \frac{a}{b} \rfloor b)y'$$

再按照 a 和 b 来归类，就可以得到

$$ax+by=ay'+(x'-\lfloor \frac{a}{b} \rfloor y')b$$

这样，$x=y',y=x'-\lfloor \frac{a}{b} \rfloor y'$ 一定会满足方程 $ax+by=\gcd(a,b)$。这样我们就构造出了它的解。

同样按照欧几里得算法的递归过程一样，到边界的时候 $b=0$，这时候整数解非常好找，就是 $x=1,y=0$。

Python实现如下：

```python
# 递归版
def ext_euclid ( a , b ):
    # ref:https://zh.wikipedia.org/wiki/扩展欧几里得算法
    if (b == 0):
        return 1, 0, a
    else:
        x1 , y1 , q = ext_euclid( b , a % b ) # q = GCD(a, b) = GCD(b, a%b)
        x , y = y1, ( x1 - (a // b) * y1 )
        return x, y, q
```

```python
# 迭代版
def egcd(a, b):
    # ref:https://blog.csdn.net/wyf12138/article/details/60476773
    if b == 0:
        return (1, 0, a)
    x, y = 0, 1
    s1, s2 = 1, 0
    r, q = a % b, a / b
    while r:
        m, n = x, y
        x = s1 - x * q
        y = s2 - y * q
        s1, s2 = m, n
        a, b = b, r
        r, q = a % b, a / b
    return (x, y, b)
```

中国剩余定理

维基百科 给出了简洁生动的说明:

用现代数学的语言来说明的话，中国剩余定理给出了以下的一元线性同余方程组：

$$(S):\ \begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

有解的判定条件，并用构造法给出了在有解情况下解的具体形式。

中国剩余定理说明：假设整数m_1，m_2，…，m_n其中任两数互质，则对任意的整数：a_1，a_2，…，a_n，方程组(S)有解，并且通解可以用如下方式构造得到：

1. 设$M = m_1 \times m_2 \times \cdots \times m_n = \prod_{i=1}^{n} m_i$是整数$m_1$，$m_2$，…，$m_n$的乘积，并设$M_i = M/m_i$，$\forall i \in \{1, 2, \cdots, n\}$，即$M_i$是除了$m_i$以外的$n-1$个整数的乘积。
2. 设$t_i = M_i^{-1}$为M_i模m_i的数论倒数：$t_i M_i \equiv 1 \pmod{m_i}$，$\forall i \in \{1, 2, \cdots, n\}$.
3. 方程组(S)的通解形式为：$x = a_1 t_1 M_1 + a_2 t_2 M_2 + \cdots + a_n t_n M_n + kM = kM + \sum_{i=1}^{n} a_i t_i M_i$，$k \in \mathbb{Z}$. 在模$M$的意义下，方程组$(S)$只有一个解：$x = \sum_{i=1}^{n} a_i t_i M_i$.

参考以上说明进行的Python实现:

```python
def CRT(mi, ai):
    # mi,ai分别表示模数和取模后的值,都为列表结构
    # Chinese Remainder Theorem
    # lcm=lambda x , y:x*y/gcd(x,y)
    # mul=lambda x , y:x*y
    # assert(reduce(mul,mi)==reduce(lcm,mi))
    # 以上可用于保证mi两两互质
    assert (isinstance(mi, list) and isinstance(ai, list))
    M = reduce(lambda x, y: x * y, mi)
    ai_ti_Mi = [a * (M / m) * gmpy2.invert(M / m, m) for (m, a) in zip(mi, ai)]
    return reduce(lambda x, y: x + y, ai_ti_Mi) % M
```

以上程序将mi当作两两互质处理,实际上有时会遇到其他情况，这时就需要逐一两两合并方程组。我参照下图实现了一个互质与不互质两种情况下都能工作良好的中国剩余定理（解同余方程组）的Python程序。

```python
def GCRT(mi, ai):
    # mi,ai分别表示模数和取模后的值,都为列表结构
    assert (isinstance(mi, list) and isinstance(ai, list))
    curm, cura = mi[0], ai[0]
    for (m, a) in zip(mi[1:], ai[1:]):
        d = gmpy2.gcd(curm, m)
        c = a - cura
        assert (c % d == 0) #不成立则不存在解
        K = c / d * gmpy2.invert(curm / d, m / d)
        cura += curm * K
        curm = curm * m / d
    return (cura % curm, curm) #(解,最小公倍数)
```

图片截自 中国剩余定理（互质与不互质的情况）。

$$\begin{cases} x = a_1 \pmod{n_1} \\ x = a_2 \pmod{n_2} \end{cases} \quad (0)$$

$$\begin{aligned} x &= n_1 k_1 + a_1 \\ x &= n_2 k_2 + a_2 \end{aligned} \quad (1)$$

$$\begin{aligned} n_1 k_1 + a_1 &= n_2 k_2 + a_2 \\ n_1 k_1 &= (a_2 - a_1) + n_2 k_2 \\ n_1 k_1 &= (a_2 - a_1) \pmod{n_2} \end{aligned}$$

显然，要想有解，必有 gcd(n1,n2)|(a2-a1)。设 gcd(n1,n2)=d，$c = a_2 - a_1$，则有：

$$\frac{n_1}{d} k_1 = \frac{c}{d} \pmod{\frac{n_2}{d}}$$

$$k_1 = \frac{c}{d} * (\frac{n_1}{d})^{-1} \pmod{\frac{n_2}{d}}$$

令 $K = \frac{c}{d} * (\frac{n_1}{d})^{-1}$，则 $k_1 = y\frac{n_2}{d} + K$，将其带入(1)式得：

$$\begin{aligned} x &= n_1 (y\frac{n_2}{d} + K) + a_1 \\ &= \frac{n_1 n_2}{d} y + n_1 K + a_1 \end{aligned}$$

即：

$$x = n_1 K + a_1 \pmod{\frac{n_1 n_2}{d}}$$

$$x = a \pmod{n} \quad (2)$$

式(2)中：

$$a = n_1 K + a_1, \quad n = \frac{n_1 n_2}{d}$$

这样，成功的将(0)式的两个方程合并为式(2)的一个方程。

最终，合并 k 个方程的最小 x 的值为 a%n。

常见攻击方式实践

准备工具

- python
  - gmpy2库
    - Windows：可从https://pypi.org/project/gmpy2/#files 直接下载已编译的安装包。
    - Linux： `sudo apt install python-gmpy2`
  - libnum库：
    - `git clone https://github.com/hellman/libnum.git && cd libnum && python setup.py install`
- yafu
  - https://sourceforge.net/projects/yafu/
- RSATool2v17.exe

RSA解密

若已知私钥d，则可以直接解密： `m=pow(c,d,n)` 。

若已知质数p和q，则通过依次计算欧拉函数值phi、私钥d可解密。简易实现如下：

```
def rsa_decrypt(e, c, p, q):
    phi = (p - 1) * (q - 1)
    n = p * q
    try:
        d = gmpy2.invert(e, phi) #求e模phi的逆
        return pow(c, d, n)
    except Exception as e:
        print "e and phi are not coprime!"
        raise e
```

在选取加密指数e时要求phi，e互质，也就是 `gcd(phi,e)==1` ，如果不满足是无法直接解密的。

为什么说这个呢？是因为有时会有乍一看有点奇怪的情况。比如SCTF2018的 `Crypto - a number problem` ，题目是

```
x**33=19260417575539056922197214220252246389137 07 mod
34364153581390166290925681987450092257732 59
tell me the smallest answer of x
```

其中 `n=34364153581390166290925681987450092257732 59` 可以直接分解得到p,q，出 `phi=(p-1)*(q-1)` ，然后惊奇地发现 `gcd(phi,33)==3` 。这时如果对加密过程比较熟悉的话，就可以想到实际上公钥 `e=11` ，明文是 `m=x^3` ，应该先求出m。然后再爆破x。

```python
for i in range(1000000):
    # 推荐使用gmpy2库运算，用pow开立方不可行
    if gmpy2.iroot(m + i * n, 3)[1]:
        x = gmpy2.iroot(m + i * n, 3)[0]
        # i==243277,x==9420391510958023
        break
```

查询已知的n的可分解情况

在线查询：https://factordb.com/

api接口：

```
curl http://factordb.com/api?query=12345
response:
{"id":"12345","status":"FF","factors":[["3",1],["5",1],["823",1]]}
```

使用yafu分解N

适用情况：p,q相差较大或较小时可快速分解。

使用方法：`yafu-x64.exe factor(233)` ， `yafu-x64.exe help`

模不互素 （`gcd(N1,N2)!=1`）

适用情况：存在两个或更多模数，且`gcd(N1,N2)!=1`。

多个模数n共用质数，则可以很容易利用欧几里得算法求得他们的质因数之一`gcd(N1,N2)`，然后这个最大公约数可用于分解模数分别得到对应的p和q，即可进行解密。实现参照本文`欧几里得算法`部分和`RSA解密`部分。

共模攻击

适用情况：明文m、模数n相同，公钥指数e、密文c不同，`gcd(e1,e2)==1`

对同一明文的多次加密使用相同的模数和不同的公钥指数可能导致共模攻击。简单证明见代码注释。

Python实现：

```python
def common_modulus(n, e1, e2, c1, c2):
    """
    ref: https://crypto.stackexchange.com/questions/16283/how-to-use-common-modulus-attack
    ∵gcd(e1,e2)==1,∴由扩展欧几里得算法，存在e1*s1+e2*s2==1
    ∴m==m^1==m^(e1*s1+e2*s2)==((m^e1)^s1)*((m^e2)^s2)==(c1^s1)*(c2^s2)
    """
    assert (libnum.gcd(e1, e2) == 1)
    _, s1, s2 = gmpy2.gcdext(e1, e2)
    # 若s1<0，则c1^s1==(c1^-1)^(-s1)，其中c1^-1为c1模n的逆元。
    m = pow(c1, s1, n) if s1 > 0 else pow(gmpy2.invert(c1, n), -s1, n)
    m *= pow(c2, s2, n) if s2 > 0 else pow(gmpy2.invert(c2, n), -s2, n)
    return m % n
```

例子：QCTF2018-XMan选拔赛 / Xman-RSA 。利用了共模攻击和模不互素。

小明文攻击

适用情况：e较小，一般为3。

公钥e很小，明文m也不大的话，于是 `m^e=k*n+m` 中的的k值很小甚至为0，爆破k或直接开三次方即可。

Python实现：

```python
def small_msg(e, n, c):
    print time.asctime(), "Let's waiting..."
    for k in xrange(200000000):
        if gmpy2.iroot(c + n * k, e)[1] == 1:
            print time.asctime(), "...done!"
            return gmpy2.iroot(c + n * k, 3)[0]
```

Rabin加密中的N可被分解

适用情况：e==2

Rabin加密是RSA的衍生算法，e==2是Rabin加密典型特征，可以百度或阅读 https://en.wikipedia.org/wiki/Rabin_cryptosystem 以了解到详细的说明，这里只关注解密方法。一般先通过其他方法分解得到p，q，然后解密。

Python实现：

```python
def rabin_decrypt(c, p, q, e=2):
    n = p * q
    mp = pow(c, (p + 1) / 4, p)
    mq = pow(c, (q + 1) / 4, q)
    yp = gmpy2.invert(p, q)
    yq = gmpy2.invert(q, p)
    r = (yp * p * mq + yq * q * mp) % n
    rr = n - r
    s = (yp * p * mq - yq * q * mp) % n
    ss = n - s
    return (r, rr, s, ss)
```

函数返回四个数，这其中只有一个是我们想要的明文，需要通过其他方式验证，当然CTF中显然就是flag字眼了。

解密方法是参照维基百科的，截图如下：

Decryption [edit]

To decode the ciphertext, the private keys are necessary.[*citation needed*] The process follows:

If c and n are known, the plaintext is then $m \in \{0, \ldots, n-1\}$ with $m^2 \equiv c \bmod n$. For a composite n (that is, like the Rabin algorithm's $n = p \cdot q$) there is no efficient method known for the finding of m. If, however n is prime (or p and q are, as in the Rabin algorithm), the Chinese remainder theorem can be applied to solve for m.

Thus the square roots

$$m_p = \sqrt{c} \bmod p$$

and

$$m_q = \sqrt{c} \bmod q$$

must be calculated (see section below).

In our example we get $m_p = 1$ and $m_q = 9$.

By applying the extended Euclidean algorithm, we wish to find y_p and y_q such that $y_p \cdot p + y_q \cdot q = 1$. In our example, we have $y_p = -3$ and $y_q = 2$.

Now, by invocation of the Chinese remainder theorem, the four square roots $+r$, $-r$, $+s$ and $-s$ of $c + n\mathbb{Z} \in \mathbb{Z}/n\mathbb{Z}$ are calculated ($\mathbb{Z}/n\mathbb{Z}$ here stands for the ring of congruence classes modulo n). The four square roots are in the set $\{0, \ldots, n-1\}$:

$$\begin{aligned} r &= (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod n \\ -r &= n - r \\ s &= (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod n \\ -s &= n - s \end{aligned}$$

One of these square roots $\bmod n$ is the original plaintext m. In our example, $m \in \{64, \mathbf{20}, 13, 57\}$.

Finding the factorization of n is possible, as Rabin pointed out in his paper, if *both* r and s can be computed, as $\gcd(|r - s|, n)$ is either p or q (where $\gcd$ means greatest common divisor). Since the greatest common divisor can be calculated efficiently, the factorization of n can be found efficiently if r and s are known. In our example (picking 57 and 13 as r and s):

$$\gcd(57 - 13, 77) = \gcd(44, 77) = 11 = q$$

Computing square roots [edit]

The decryption requires to compute square roots of the ciphertext c modulo the primes p and q. Choosing $p \equiv q \equiv 3 \bmod 4$ allows to compute square roots more easily by

$$m_p = c^{\frac{1}{4}(p+1)} \bmod p$$

and

$$m_q = c^{\frac{1}{4}(q+1)} \bmod q.$$

Wiener's Attack

适用情况：e过大或过小。

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

在e过大或过小的情况下，可使用算法从e中快速推断出d的值。详细的算法原理可以阅读：低解密指数攻击 。

```python
from Crypto.PublicKey import RSA
import ContinuedFractions, Arithmetic

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
```

私钥文件修复

适用情况：提供破损的私钥文件。

例题：Jarvis OJ-God Like RSA

参考 https://www.40huo.cn/blog/rsa-private-key-recovery-and-oaep.html 修复存储私钥的文件，得到p和q。

LSB Oracle Attack

适用情况：可以选择密文并泄露最低位。

在一次RSA加密中，明文为m，模数为n，加密指数为e，密文为c。我们可以构造出 `c'=((2^e)*c)%n=((2^e)*(m^e))%n=((2*m)^e)%n` ， 因为m的两倍可能大于n，所以经过解密得到的明文是 `m'=(2*m)%n` 。我们还能够知道 `m'` 的最低位 `lsb` 是1还是0。 因为n是奇数，而 `2*m` 是偶数，所以如果 `lsb` 是0，说明 `(2*m)%n` 是偶数，没有超过n，即 `m<n/2.0` ，反之则 `m>n/2.0` 。举个例子就能明白 `2%3=2` 是偶数，而 `4%3=1` 是奇数。以此类推，构造密文 `c"=(4^e)*c)%n` 使其解密后为 `m"=(4*m)%n` ，判断 `m"` 的奇偶性可以知道 `m` 和 `n/4` 的大小关系。所以我们就有了一个二分算法，可以在对数时间内将m的范围逼近到一个足够狭窄的空间。

更多信息可参考：RSA Least-Significant-Bit Oracle Attack 和 RSA least significant bit oracle attack 。

Python实现：

```python
import decimal
def oracle():
    return lsb == 'odd'


def partial(c, e, n):
    k = n.bit_length()
    decimal.getcontext().prec = k  # for 'precise enough' floats
    lo = decimal.Decimal(0)
    hi = decimal.Decimal(n)
    for i in range(k):
        if not oracle(c):
            hi = (lo + hi) / 2
        else:
            lo = (lo + hi) / 2
        c = (c * pow(2, e, n)) % n
        # print i, int(hi - lo)
    return int(hi)
```

例子：QCTF2018-XMan选拔赛/Baby RSA

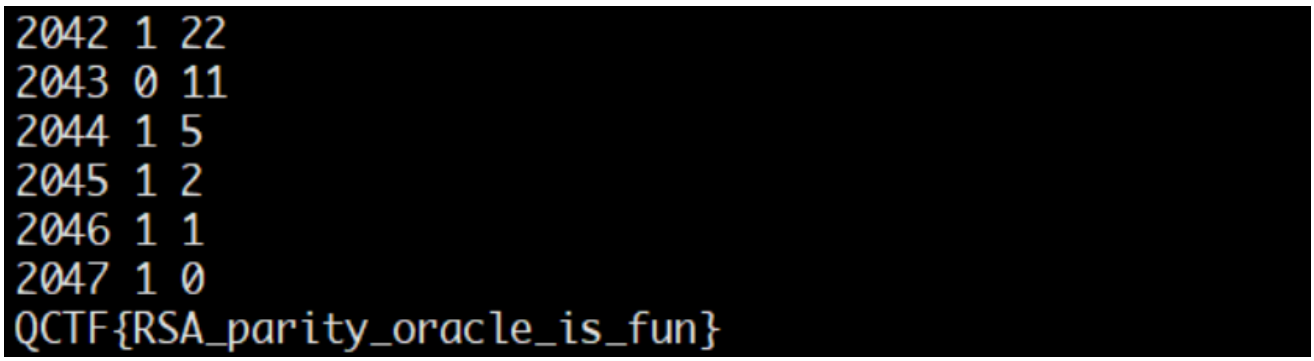
题目如下

```
e = 0x10001
n =
0x0b765daa79117afe1a77da7ff8122872bbcbddb322bb078fe0786dc40c9033fadd639adc48c3f2627fb7cb59bb
0658707fe516967464439bdec2d6479fa3745f57c0a5ca255812f0884978b2a8aaeb750e0228cbe28a1e5a63bf03
09b32a577eecea66f7610a9a4e720649129e9dc2115db9d4f34dc17f8b0806213c035e22f2c5054ae584b440def0
0afbccd458d020cae5fd1138be6507bc0b1a10da7e75def484c5fc1fcb13d11be691670cf38b487de9c4bde6c2c6
89be5adab08b486599b619a0790c0b2d70c9c461346966bcbae53c5007d0146fc520fa6e3106fbfc899052207788
70a7119831c17f98628563ca020652d18d72203529a784ca73716db
c =
0x4f377296a19b3a25078d614e1c92ff632d3e3ded772c4445b75e468a9405de05d15c77532964120ae11f8655b6
8a630607df0568a7439bc694486ae50b5c0c8507e5eecdea4654eeff3e75fb8396e505a36b0af40bd5011990663a
7655b91c9e6ed2d770525e4698dec9455db17db38fa4b99b53438b9e09000187949327980ca903d0eef114afc42b
771657ea5458a4cb399212e943d139b7ceb6d5721f546b75cd53d65e025f4df7eb8637152ecbb6725962c7f66b71
4556d754f41555c691a34a798515f1e2a69c129047cb29a9eef466c206a7f4dbc2cea1a46a39ad3349a7db56c1c9
97dc181b1afcb76fa1bbbf118a4ab5c515e274ab2250dba1872be0

λ nc 47.96.239.28 23333
---------------------------- baby rsa -----------------------------
Come and Decode your data
If you give me ciphertext, I can tell you whether decoded data is even or odd
You can input ciphertext(hexdecimal) now
1
odd
```

解题脚本：

```python
# -*- coding: utf-8 -*-
# by https://findneo.github.io/
# ref:
# https://crypto.stackexchange.com/questions/11053/rsa-least-significant-bit-oracle-attack
# https://ctf.rip/sharif-ctf-2016-lsb-oracle-crypto-challenge/
# https://introspelliam.github.io/2018/03/27/crypto/RSA-Least-Significant-Bit-Oracle-Attack/
import libnum, gmpy2, socket, time, decimal


def oracle(c1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    hostname = '47.96.239.28'
    port = 23333
    s.connect((hostname, port))
    s.recv(1024)
    s.send(hex(c1)[2:].strip("lL") + '\n')
    res = s.recv(1024).strip()
    s.close()
    if res == 'even': return 0
    if res == 'odd':
        return 1
    else:
        assert (0)


def partial(c, n):
```

```
    global c_of_2
    k = n.bit_length()
    decimal.getcontext().prec = k  # allows for 'precise enough' floats
    lower = decimal.Decimal(0)
    upper = decimal.Decimal(n)
    for i in range(k):
        possible_plaintext = (lower + upper) / 2
        # lower==0 when i<1809
        flag = oracle(c)
        if not flag:
            upper = possible_plaintext  # plaintext is in the lower half
        else:
            lower = possible_plaintext  # plaintext is in the upper half
        c = (c * c_of_2) % n  # multiply y by the encryption of 2 again
        print i, flag, int(upper - lower)
        # time.sleep(0.2)
    # By now, our plaintext is revealed!
    return int(upper)


def main():
    print "[*] Conducting Oracle attack..."
    return partial((c * c_of_2) % n, n)


if __name__ == '__main__':
    e = 0x10001
    n =
0x0b765daa79117afe1a77da7ff8122872bbcbddb322bb078fe0786dc40c9033fadd639adc48c3f2627fb7cb59bb
0658707fe516967464439bdec2d6479fa3745f57c0a5ca255812f0884978b2a8aaeb750e0228cbe28a1e5a63bf03
09b32a577eecea66f7610a9a4e720649129e9dc2115db9d4f34dc17f8b0806213c035e22f2c5054ae584b440def0
0afbccd458d020cae5fd1138be6507bc0b1a10da7e75def484c5fc1fcb13d11be691670cf38b487de9c4bde6c2c6
89be5adab08b486599b619a0790c0b2d70c9c461346966bcbae53c5007d0146fc520fa6e3106fbfc899052207788
70a7119831c17f98628563ca020652d18d72203529a784ca73716db
    c =
0x4f377296a19b3a25078d614e1c92ff632d3e3ded772c4445b75e468a9405de05d15c77532964120ae11f8655b6
8a630607df0568a7439bc694486ae50b5c0c8507e5eecdea4654eeff3e75fb8396e505a36b0af40bd5011990663a
7655b91c9e6ed2d770525e4698dec9455db17db38fa4b99b53438b9e09000187949327980ca903d0eef114afc42b
771657ea5458a4cb399212e943d139b7ceb6d5721f546b75cd53d65e025f4df7eb8637152ecbb6725962c7f66b71
4556d754f41555c691a34a798515f1e2a69c129047cb29a9eef466c206a7f4dbc2cea1a46a39ad3349a7db56c1c9
97dc181b1afcb76fa1bbbf118a4ab5c515e274ab2250dba1872be0
    c_of_2 = pow(2, e, n)
    m = main()
    # m = 560856645743734814774953158390773525781916094468093308691660509501812349
    print libnum.n2s(m)
    # QCTF{RSA_parity_oracle_is_fun}
```



选择密文攻击

适用情况：可以构造任意密文并获得对应明文。

这个好理解，在一个RSA加密过程中，明文为m，密文为c，模数为n，加密指数为e，选取x以满足`gcd(x,n)==1`从而使x模n的逆存在，构造密文`c'=c*(x^e)`使解密后明文为`m'=(m*x)%n`，则`m=m'*x^-1(mod n)`。可参看`模意义下的运算法则部分`。

广播攻击

适用情况：模数n、密文c不同，明文m、加密指数e相同。一般会是e=k，然后给k组数据

使用不同的模数n，相同的公钥指数e加密相同的信息。就会得到多个(m^e) ==ci (mod ni)，将(m^e)视为一个整体M，这就是典型的中国剩余定理适用情况。按照本文的`中国剩余定理`小节容易求得m^e的值，当e较小时直接开e方即可，可使用`gmpy2.iroot(M,e)`方法。

Python实现：参见本文`中国剩余定理`小节。

后话

RSA可谓现代密码学的中流砥柱，关于它的可行攻击方法研究还有很多，诸如Timing Attack，Padding oracle attack，Side-channel analysis attacks等类型的攻击，本文仅介绍了一些通俗易懂的方法，读者还可以阅读 CTF wiki中的非对称加密部分，以及以 RSA (cryptosystem) 为目录结合谷歌进行进一步学习。

本文的例题附件、代码段、工具和后续更新都会放在 RSA-ATTACK，欢迎 star & watch 。

参考链接

Practical Padding Oracle Attacks on RSA

CTF wiki中的非对称加密部分