

攻防世界 · Normal_RSA · wp

原创

connamon 于 2022-03-31 23:49:47 发布 53 收藏

文章标签: [网络安全](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/sml918451478/article/details/123886833>

版权

Normal RSA

95 最佳Writeup由露思提供

WP 建议

难度系数: ★★★★★ 5.0

题目来源: PCTF

题目描述: 你和小鱼走走走啊走, 走到下一个题目一看你一愣, 怎么还是一个数学题啊 小鱼一笑, hhhh数学在密码学里面很重要的! 现在知道吃亏了吧! 你哼一声不服气, 我知道数学 很重要了! 但是工具也很重要, 你看我拿工具把他解出来! 你打开电脑折腾了一会还真的把答案 做了出来, 小鱼有些吃惊, 向你投过来一个赞叹的目光

题目场景: 暂无

题目附件: 附件1

flag..

提交

CSDN @connamon

附件情况

enc后缀: 用于通过在UUenconded格式文件, 它们是加密的文

pem后缀: PEM文件是用于对安全网站进行身份验证的Base64编码的证书文件, 它可能包含私钥、证书颁发机构(CA)服务器证书或组成信任链的其他各种证书。PEM文件通常从基于Linux的Apache或Nginx Web服务器中导入, 并且与OpenSSL应用程序兼容。

名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
flag.enc	32	32	ENC 文件	2016/4/29 17:...	C63C8100
pubkey.pem	138	125	PEM 文件	2016/4/29 17:...	8216979D

CSDN @connamon

```
pubkey.pem - 记事本
文件 编辑 查看

-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgMBAAE=
-----END PUBLIC KEY-----

CSDN @connamon
```

```
flag.enc - 记事本
文件 编辑 查看

m>愤#钺訃口竟x燻?渊?口Im越劫?
y

CSDN @connamon
```

发现有base64编码，但是并没有什么办法解码

发现flag文件没有办法打开，那就用记事本、openssl打开pubkey文件

```
openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
```

```
OpenSSL> rsa -pubin -text -modulus -in pubkey.pem
RSA Public-Key: (256 bit)
Modulus:
 00:c2:63:6a:e5:c3:d8:e4:3f:fb:97:ab:09:02:8f:
 1a:ac:6c:0b:f6:cd:3d:70:eb:ca:28:1b:ff:e9:7f:
 be:30:dd
Exponent: 65537 (0x10001)
Modulus=C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgMBAAE=
-----END PUBLIC KEY-----
OpenSSL>

CSDN @connamon
```

e=65537

n=p*q=C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD（十六进制）

n=10866948760844599168252082612378495977388271279679231539839049698621994994673（十进制）

利用素数分解分解出p和q

p=275127860351348928173285174381581152299

q=319576316814478949870590164193048041239

素因子分解工具: factordb.com

Search Sequences Report results Factor tables Status Downloads Login

87924348264132406875276140514499937145050893665602592992418171647042491658461 Factorize!

Result:

status (2)	digits	number
FF	77 (show)	8792434826...61 <77> = 275127860351348928173285174381581152299 <39> · 319576316814478949870590164193048041239 <39>

More information

ECM

CSDN @connamon

利用rsatool计算d

d=10866948760844599168252082612378495977388271279679231539839049698621994994673

rsa工具下载请查看buuctf·rsarsa·wp文章

RSA-Tool 2 by tE!

Keysize (Bits): 256

Number Base: 10

Random data generation: Start 00000000 0%

Public Exponent (E) [HEX]: 10001

1st Prime (P): 275127860351348928173285174381581152299

2nd Prime (Q): 319576316814478949870590164193048041239

Modulus (N) [R]: 87924348264132406875276140514499937145050893665602592992418171647042491658461

Exact size: 0 Bits

Private Exponent (D): 10866948760844599168252082612378495977388271279679231539839049698621994994673

Generate Test

Calc. D Factor N

Help Exit

Factoring info (Prime factors): 0

Use MPQS method only No time checks

Ready. To create RSA Keys, press >Start< now to generate some random data...

CSDN @connamon

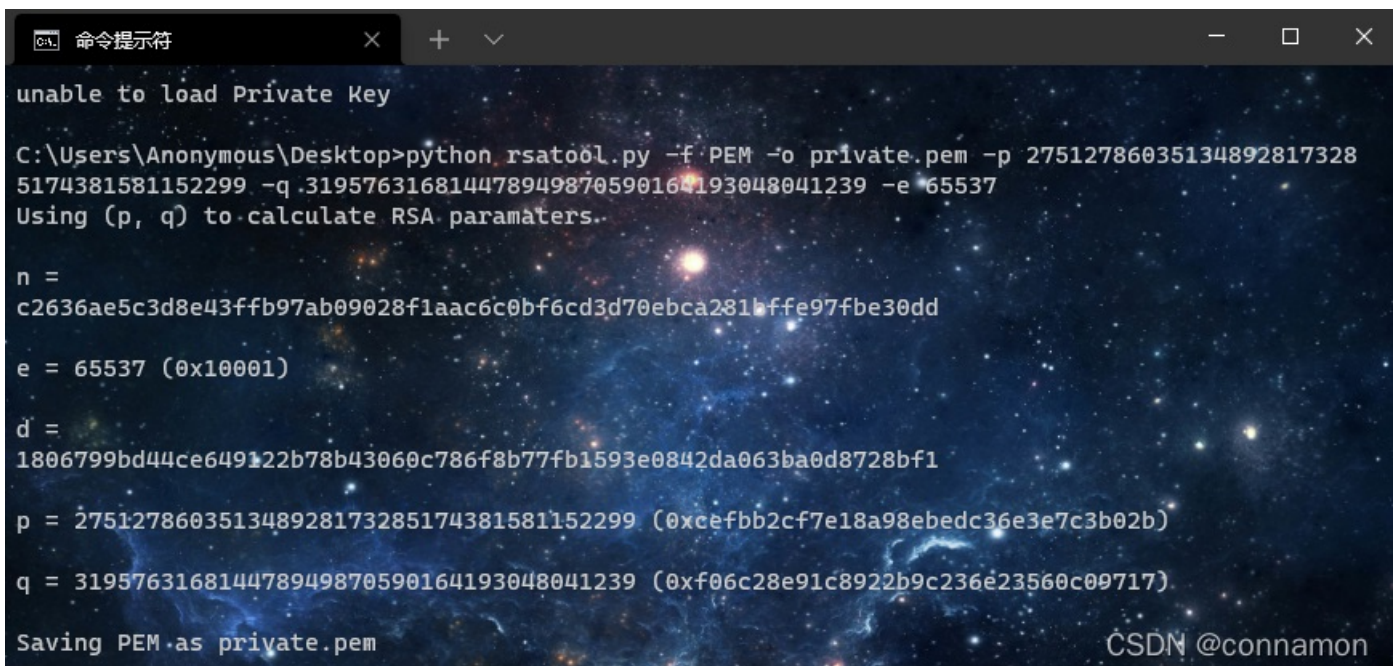
解出了之后发现没有密文，换条思路，查看一下wp

下载一个工具rsatool.py脚本

运行脚本之后在终端输入命令，需要先安装gmpy2

```
pip install gmpy2
```

```
python rsatool.py -f PEM -o private.pem -p 275127860351348928173285174381581152299 -q 319576316814478949870
```



```
命令提示符
unable to load Private Key

C:\Users\Anonymous\Desktop>python rsatool.py -f PEM -o private.pem -p 27512786035134892817328
5174381581152299 -q 319576316814478949870590164193048041239 -e 65537
Using (p, q) to calculate RSA paramaters.

n =
c2636ae5c3d8e43ffb97ab09028f1aac6c0bf6cd3d70ebca281bffe97f97be30dd

e = 65537 (0x10001)

d =
1806799bd44ce649122b78b43060c786f8b77fb1593e0842da063ba0d8728bf1

p = 275127860351348928173285174381581152299 (0xcefb2cf7e18a98ebdc36e3e7c3b02b)

q = 319576316814478949870590164193048041239 (0xf06c28e91c8922b9c236e23560c09717)

Saving PEM as private.pem

CSDN @connamon
```

即可生成private.pem文件

最后用其解开flag.enc文件，在终端输入：

```
openssl rsautl -decrypt -in flag.enc -inkey private.pem
```



```
命令提示符
C:\Users\Anonymous\Desktop>openssl rsautl -decrypt -in flag.enc -inkey private.pem

CSDN @connamon
```

rsatool脚本 [rsatool/rsatool.py at master · ius/rsatool \(github.com\)](https://github.com/ius/rsatool)

```
#!/usr/bin/env python3
import base64
import argparse
import random
import sys
import textwrap

import gmpy2

from pyasn1.codec.der import encoder
from pyasn1.type.univ import Sequence, Integer

PEM_TEMPLATE = (
    '-----BEGIN RSA PRIVATE KEY-----\n'
    '%s\n'
    '-----END RSA PRIVATE KEY-----\n'
)

DEFAULT_EXP = 65537
```

```

def factor_modulus(n, d, e):
    """
    Efficiently recover non-trivial factors of n
    See: Handbook of Applied Cryptography
    8.2.2 Security of RSA -> (i) Relation to factoring (p.287)
    http://www.cacr.math.uwaterloo.ca/hac/
    """
    t = e * d - 1
    s = 0

    if 17 != gmpy2.powmod(17, e * d, n):
        raise ValueError("n, d, e don't match")

    while True:
        quotient, remainder = divmod(t, 2)

        if remainder != 0:
            break

        s += 1
        t = quotient

    found = False

    while not found:
        i = 1
        a = random.randint(1, n - 1)

        while i <= s and not found:
            c1 = pow(a, pow(2, i - 1, n) * t, n)
            c2 = pow(a, pow(2, i, n) * t, n)

            found = c1 != 1 and c1 != (-1 % n) and c2 == 1

            i += 1

    p = gmpy2.gcd(c1 - 1, n)
    q = n // p

    return p, q

class RSA:
    def __init__(self, p=None, q=None, n=None, d=None, e=DEFAULT_EXP):
        """
        Initialize RSA instance using primes (p, q)
        or modulus and private exponent (n, d)
        """

        self.e = e

        if p and q:
            assert gmpy2.is_prime(p), 'p is not prime'
            assert gmpy2.is_prime(q), 'q is not prime'

            self.p = p
            self.q = q
        elif n and d:

```

```

        self.p, self.q = factor_modulus(n, d, e)
    else:
        raise ValueError('Either (p, q) or (n, d) must be provided')

    self._calc_values()

def _calc_values(self):
    self.n = self.p * self.q

    if self.p != self.q:
        phi = (self.p - 1) * (self.q - 1)
    else:
        phi = (self.p ** 2) - self.p

    self.d = gmpy2.invert(self.e, phi)

    # CRT-RSA precomputation
    self.dP = self.d % (self.p - 1)
    self.dQ = self.d % (self.q - 1)
    self.qInv = gmpy2.invert(self.q, self.p)

def to_pem(self):
    """
    Return OpenSSL-compatible PEM encoded key
    """
    b64 = base64.b64encode(self.to_der()).decode()
    b64w = "\n".join(textwrap.wrap(b64, 64))
    return (PEM_TEMPLATE % b64w).encode()

def to_der(self):
    """
    Return parameters as OpenSSL compatible DER encoded key
    """
    seq = Sequence()

    for idx, x in enumerate(
        [0, self.n, self.e, self.d, self.p, self.q, self.dP, self.dQ, self.qInv]
    ):
        seq.setComponentByPosition(idx, Integer(x))

    return encoder.encode(seq)

def dump(self, verbose):
    vars = ['n', 'e', 'd', 'p', 'q']

    if verbose:
        vars += ['dP', 'dQ', 'qInv']

    for v in vars:
        self._dumpvar(v)

def _dumpvar(self, var):
    val = getattr(self, var)

    def parts(s, n):
        return '\n'.join([s[i:i + n] for i in range(0, len(s), n)])

    if len(str(val)) <= 40:
        print('%s = %d (%#x)\n' % (var, val, val))
    else:

```

```

    print('%s =' % var)
    print(parts('%x' % val, 80) + '\n')

if __name__ == '__main__':
    parser = argparse.ArgumentParser()

    parser.add_argument('-n', type=lambda x: int(x, 0),
                        help='modulus. format : int or 0xhex')
    parser.add_argument('-p', type=lambda x: int(x, 0),
                        help='first prime number. format : int or 0xhex')
    parser.add_argument('-q', type=lambda x: int(x, 0),
                        help='second prime number. format : int or 0xhex')
    parser.add_argument('-d', type=lambda x: int(x, 0),
                        help='private exponent. format : int or 0xhex')
    parser.add_argument('-e', type=lambda x: int(x, 0),
                        help='public exponent (default: %d). format : int or 0xhex' %
                             DEFAULT_EXP, default=DEFAULT_EXP)
    parser.add_argument('-o', '--output', help='output filename')
    parser.add_argument('-f', '--format', choices=['DER', 'PEM'], default='PEM',
                        help='output format (DER, PEM) (default: PEM)')
    parser.add_argument('-v', '--verbose', action='store_true', default=False,
                        help='also display CRT-RSA representation')

    args = parser.parse_args()

    if args.p and args.q:
        print('Using (p, q) to calculate RSA parameters\n')
        rsa = RSA(p=args.p, q=args.q, e=args.e)
    elif args.n and args.d:
        print('Using (n, d) to calculate RSA parameters\n')
        rsa = RSA(n=args.n, d=args.d, e=args.e)
    else:
        parser.print_help()
        parser.error('Either (p, q) or (n, d) needs to be specified')

    if args.format == 'DER' and not args.output:
        parser.error('Output filename (-o) required for DER output')

    rsa.dump(args.verbose)

    if args.format == 'PEM':
        data = rsa.to_pem()
    elif args.format == 'DER':
        data = rsa.to_der()

    if args.output:
        print('Saving %s as %s' % (args.format, args.output))

        fp = open(args.output, 'wb')
        fp.write(data)
        fp.close()
    else:
        sys.stdout.buffer.write(data)

```